TESIS DOCTORAL

OBJETOS LOCALMENTE EFECTIVOS Y TIPOS ABSTRACTOS DE DATOS

Vico Pascual Martínez-Losa

UNIVERSIDAD DE LA RIOJA

TESIS DOCTORAL

OBJETOS LOCALMENTE EFECTIVOS
Y
TIPOS ABSTRACTOS DE DATOS

Vico Pascual Martínez-Losa

Universidad de La Rioja
Servicio de Publicaciones
2003

Esta tesis doctoral, dirigida por los doctores D. Laureano Lambán Pardo y D. Julio Rubio García fue leida el 9 de Abril de 2002, y obtuvo la calificación de Sobresaliente cum Laude por unanimidad

Edita: Universidad de La Rioja
Servicio de Publicaciones

ISBN 84-688-0904-7

UNIVERSIDAD DE LA RIOJA

Departamento de Matemáticas y Computación

OBJETOS LOCALMENTE EFECTIVOS Y TIPOS ABSTRACTOS DE DATOS

TESIS DOCTORAL

Vico Pascual Martínez-Losa

Logroño, Enero de 2002

UNIVERSIDAD DE LA RIOJA

Departamento de Matemáticas y Computación

TESIS DOCTORAL

OBJETOS LOCALMENTE EFECTIVOS Y TIPOS ABSTRACTOS DE DATOS

Presentada por

Vico Pascual Martínez-Losa

para la obtención del título de Doctora en Matemáticas

Directores: Dr. D. Laureano Lambán Pardo

Dr. D. Julio Rubio García

Logroño, Enero de 2002

Agradecimientos

A Óscar

A Teresa, Antonio y Roberto

Esta tesis es el resultado de varios años de trabajo en los que muchas personas han estado ahí, a mi lado. En primer lugar, mis directores, Julio y Laure, siempre al pie del cañón. Debo agradecerles su total disponibilidad, sus ideas brillantes y su paciente y constante supervisión. Laure siempre pendiente en el despacho contiguo. Julio, en estos dos últimos años paseando entre su despacho y el mío y, hasta entonces en Zaragoza, siempre alerta de mis mensajes de correo electrónico y de mis llamadas de teléfono. Han conseguido que nunca me sintiera sola con este trabajo. También quiero agradecer su trato personal y sus ánimos en los momentos duros, al igual que los de César, Jesús Mari y Juanjo.

Otras personas también han contribuido a que este trabajo saliera adelante. Aún a riesgo de dejarme alguna de ellas: Ángel Luis y Francisco por su ayuda en los problemas técnicos cuando una aplicación no funcionaba o cuando un fichero no se imprimía, a Luis por sus clases particulares de Lógica y Categorías, a Juan Luis por su ayuda con el LaTeX, y, al grupo Nóesis por su caluroso trato en Zaragoza.

No puedo olvidarme de mis padres y hermano, siempre pendientes de la evolución de esta tesis y ayudando en lo que hiciera falta. Y, muy especialmente, de Óscar, por su constante ánimo, ayuda y apoyo.
A ellos y a todos los que siempre han estado ahí, gracias.*

*También al Departamento de Matemáticas y Computación por el ambiente agradable en el que hemos desarrollado este trabajo así como por el uso de sus medios, a la Universidad de La Rioja por sus ayudas y bolsas de viaje para la realización de esta tesis y a la DGES por su financiación a través del proyecto PB98-1621-C02-01.

Índice General

Introducción

EAT (siglas de *Effective Algebraic Topology*) es un conjunto de programas Common Lisp, escrito bajo la dirección de Sergeraert y dedicado al Cálculo Simbólico en Topología Algebraica. El sistema EAT es la primera plasmación real (ejecutable sobre un computador) de ideas de Sergeraert como la *homología efectiva* ([112], [99], [102]), la *codificación funcional* [113] y los *objetos localmente efectivos* [101]. Estas ideas, así como los nuevos algoritmos de cálculo basados en ellas, han dotado a EAT de una gran potencia de cálculo que le han permitido obtener resultados (en concreto, grupos de homología) que no han podido ser confirmados (¡ni refutados!) por ningún otro método (teórico o automático); algunos ejemplos pueden encontrarse en [95] o [105].

En este contexto, parecía interesante realizar un análisis formal del programa que, aún en el caso de que no pudiese dar lugar a una prueba completa de su corrección, al menos permitiese razonar sobre los procesos internos de cálculo. Con este objetivo en mente, se consideró que las técnicas de *especificación algebraica* (véanse, por ejemplo [40] y [72]) podrían ofrecer los recursos necesarios para emprender esta tarea. La elección de esta herramienta, frente a otros métodos formales en Informática (véanse varias alternativas, por ejemplo, en [84], [3] y [18]), estuvo basada en tres puntos. En primer lugar, se trata de una de las técnicas formales más antiguas (véanse [124], [52], [49] y [53]), a la que se han dedicado más investigadores (véanse, solo como ejemplo, la serie de *Workshops on Algebraic Development Techniques*, *WADT*, [12], [42], [86], entre otros) y que, por tanto, dispone de más experiencias y tecnología para la especificación de sistemas [6]. En segundo lugar, la especificación algebraica está muy vinculada con (aunque no se restringe a) los *tipos abstractos de datos* y a la hora de comenzar a estudiar un sistema complejo como EAT parece oportuno comenzar por sus módulos más elementales: sus estructuras de datos. Y en tercer y último lugar, la especificación algebraica utiliza como lenguajes subyacentes los del Álgebra Universal y la Teoría de Categorías, que son obviamente muy adecuados en el contexto del Cálculo Simbólico en Topología Algebraica: los objetos a manipular son estructuras algebraicas (grupos graduados, complejos de cadenas, etc.) y, por otra parte, el nacimiento mismo de la Teoría de Categorías estuvo muy ligado al desarrollo de la Topología Algebraica (alrededor de los años 50; véase [41]).

Una vez elegido el marco en el que desarrollar la especificación de EAT (el de las especificaciones algebraicas) conviene hacer notar dos características diferenciales de nuestro problema. Por una parte, las estructuras de datos de EAT codifican conjuntos de naturaleza infinita, por medio de técnicas de programación funcional (a través de los conceptos de Sergeraert ya mencionados: la codificación funcional y los objetos localmente efectivos). La mayoría de los esfuerzos en especificación algebraica han estado sin embargo ligados al análisis de estructuras finitas o finitamente generables, por medio del concepto de semántica inicial o de sus variantes; ver, por ejemplo [40]. (De hecho, algunos autores [29] consideran precisamente esta característica como esencial en la metodología de diseño basada en Tipos Abstractos de Datos.)

Por otra parte, la especificación algebraica suele ser presentada como una técnica útil en las fases *tempranas* (*early*, en inglés) del desarrollo de un sistema informático. Hablando sin rigor, a la fase de especificación (algebraica) le seguiría un proceso de diseño detallado para terminar con la implementación real del sistema. La especificación algebraica proporcionaría así una referencia, matemáticamente consistente, contra la que debería validarse la corrección de cada una de las siguientes fases en el desarrollo del sistema. Nuestro problema es completamente diferente (incluso es, en un cierto sentido, el opuesto). Partimos de un sistema previamente desarrollado y que lleva en funcionamiento varios años (al menos desde 1991), con cierto éxito en su campo de aplicación. Por tanto, nuestro objetivo no es influir en el diseño de dicho programa, sino más bien obtener modelos matemáticos que nos permitan razonar sobre sus procesos internos de cálculo. Resumimos esta situación diciendo que las especificaciones algebraicas no son utilizadas en nuestro caso para el *diseño* de un sistema (como es habitual) sino para la *modelización* de un sistema. Una consecuencia de ello es que el lenguaje de programación, que en la mayoría de la literatura sobre especificaciones algebraicas no deja de ser un referente lejano (incluso cuando se habla de implementaciones [39]), pasa a ser un elemento de primera importancia y muy concreto (Common Lisp en nuestro caso) en la modelización.

Bien sea por estas dos razones, o bien porque la especificación algebraica no haya sido confrontada con sistemas tan complejos (por tamaño y conceptualmente) como EAT (al menos la literatura consultada solo recoge detalladamente ejemplos académicos, pese a que se citan con frecuencia en las introducciones casos de uso industrial de las especificaciones algebraicas), resultó que, lejos de tratarse de una simple aplicación de técnicas conocidas, nuestro problema se convirtió en un verdadero desafío, en el que ha sido necesario introducir nuevas construcciones, conceptos y técnicas para alcanzar los objetivos propuestos. A presentar estas novedades está dedicada esta memoria.

Hablando informalmente, entre las estructuras de datos de EAT pueden distinguirse dos *capas*. Una primera, básica, constituida por datos *estándar* (como listas finitas, vectores de enteros o símbolos, etc.) y otra segunda, funcional, dedicada a codificar estructuras algebraicas (grupos, módulos, etc.), cuyos elementos son datos de la primera capa. Para las estructuras de datos de la primera capa la aproximación clásica de las especificaciones algebraicas (simplificando, la semántica inicial) es suficiente, no tratándose por tanto de una tarea de investigación. (La especificación detallada se vería dificultada por el hecho de que, en esa capa básica, se utilizan por razones de eficiencia efectos laterales, como modificaciones destructivas de listas o vectores; pero esas dificultades son bien conocidas y se consideran "resueltas" en la literatura. Indiquemos, de pasada, que la segunda capa de estructuras de datos está libre de efectos laterales y que, por tanto, para nuestro trabajo podemos considerar que EAT es un sistema basado en programación funcional *pura*.)

Las estructuras de datos de la segunda capa, por el contrario, no se adecúan a la semántica inicial. Esta observación fue ya señalada en [69], donde también se indicó que las estructuras de datos de EAT (segunda capa) eran implementaciones de tipos abstractos que eran *canónicos* en un sentido que en aquel trabajo no era precisado en exceso. Una forma de entender esta memoria es verla como la búsqueda precisa y formal de en qué sentido las estructuras de EAT son "canónicas". La solución, presentada en esta memoria (y previamente en [67]), consiste en la demostración de que dichas estructuras son *universales* en el sentido de la Teoría de Categorías. Concretamente, forman parte de objetos finales en ciertas categorías de implementaciones de tipos abstractos de datos.

Se ve por tanto que no solamente el punto de vista *inicial* (en el sentido de la Teoría de Categorías) no es adecuado, sino que la perspectiva conveniente es la relacionada con los objetos

finales. Tras una aparición temprana en la historia de las especificaciones algebraicas [121], no se prestó excesiva atención a la semántica final, hasta que, sobre todo en la última década, ha vuelto con fuerza relacionada con la especificación de sistemas orientados a objetos. Destacan en este campo las especificaciones ocultas ([46], [23], [45], [27], [76], [94] y [47]) y los métodos coalgebraicos ([91], [106], [60], [61], [62], [76] y [27]). Resulta ligeramente sorprendente ver al sistema EAT, basado en la programación funcional, emparentado con los formalismos orientados a objetos. Sin embargo, tanto en [68], como con más detalle en el capítulo 2 de esta memoria, mostramos que la relación es mucho más natural de lo que pudiese parecer.

Finalmente, otro modo de entender este trabajo consiste en verlo como una interpretación de los objetos localmente efectivos de Sergeraert (así como de sus diferencias con los objetos efectivos) en el marco de las implementaciones de los tipos abstractos de datos. Como parte de esa interpretación aparece la justificación del uso de dichos objetos para la codificación de estructuras de datos infinitas que, como hemos indicado, es una de las características relevantes de EAT. (Esto muestra otra relación con los métodos coalgebraicos, que han sido propuestos frecuentemente en la literatura [91] como alternativa para la especificación de estructuras de datos infinitas.)

Este texto está organizado como se explica a continuación. El primer capítulo comienza con una sección en la que se presenta brevemente el sistema EAT. En la segunda sección se fijan las definiciones, terminología y notaciones relativas a la Teoría de Conjuntos, la Teoría de Categorías y las especificaciones algebraicas que se usan en este trabajo. El capítulo 2 presenta la construcción central de la memoria, denotada $()_{imp}$, en un contexto puramente algebraico. Además se interpreta la construcción $()_{imp}$ en el marco de la Teoría de Categorías y se analizan detalladamente sus relaciones con las especificaciones ocultas y las coalgebraicas. En el capítulo 3 se introduce la categoría de implementaciones de un tipo abstracto de datos (en Common Lisp) y se muestra el papel de la construcción $()_{imp}$ para interpretar en este contexto las estructuras de EAT en términos de objetos finales de categorías. En el último capítulo, el cuarto, los resultados anteriores son refinados al incluir explícitamente información relativa a los invariantes y las igualdades de las implementaciones. Allí se explican también las diferencias entre las representaciones localmente efectivas y las efectivas. La memoria termina con un apartado de conclusiones y trabajo futuro y la bibliografía.

Capítulo 1

Preliminares

1.1 Presentación de EAT

El propósito de esta sección es presentar el sistema EAT a través de ejemplos que nos permitan estudiar algunas de sus características. Como ya hemos comentado en la introducción, el sistema EAT está dedicado al cálculo en Topología Algebraica. Ésta se ocupa de asociar invariantes algebraicos a espacios topológicos, para estudiar ciertas propiedades de los espacios topológicos a través de estos invariantes algebraicos. (El argumento para buscar invariantes algebraicos de espacios topológicos es que los objetos algebraicos son más fáciles de identificar y clasificar que los topológicos.) Ejemplos de invariantes algebraicos asociados a espacios topológicos son, entre otros, los grupos de homología y los de homotopía. Concretamente EAT está dedicado al cálculo de los grupos de homología de espacios de lazos iterados.

El primer paso para trabajar en Topología Algebraica utilizando un ordenador es dar un modelo combinatorial del espacio topológico a estudiar. Un modelo combinatorial está formado por símbolos y funciones entre ellos, de ahí que el Cálculo Simbólico sea adecuado para manipularlo. Uno de los principales problemas que aparecen es que los espacios pueden ser de naturaleza infinita, en el sentido de que no admitan una descripción combinatorial en términos de un número finito de elementos, y sin embargo puede ocurrir que sus invariantes sean de tipo finito y, por tanto, susceptibles de ser calculados por medio de un ordenador. En este sentido, la primera dificultad que aparece es la manipulación en el ordenador de representaciones de espacios de naturaleza infinita.

La teoría de la Homología Efectiva (véanse [112], [114]), debida a Sergeraert, es un marco en el que abordar el problema de la calculabilidad en Topología Algebraica, y es la teoría en la que reposan los algoritmos de cálculo de EAT. Utilizando las técnicas introducidas por Sergeraert, en [95] se estudió el problema concreto del cálculo de la homología de espacios de lazos iterados, estudio que dio lugar al sistema EAT. La noción clave en todo ello es la de *objeto con homología efectiva* . La idea es enriquecer los objetos (por ejemplo, los modelos combinatoriales de espacios topológicos) con información adicional de forma que se pueda aplicar sobre ellos un algoritmo de cálculo de su homología, incluso en el caso de que los objetos de partida no sean de tipo finito.

Mostramos a continuación algunos ejemplos de cálculo del sistema EAT. Pese a ser un ejemplo con resultado bien conocido, vamos a comenzar calculando el grupo de homología $H_3(S^3)$, es decir, el tercer grupo de homología de la esfera S^3. Las esferas, en cualquier dimensión,

poseen una versión combinatoria por medio de conjuntos simpliciales, siendo éstos unos de los modelos combinatoriales más utilizados [78]. En EAT* construimos la esfera de dimensión 3 tal y como se muestra a continuación.

```
> (setf s3 (sphere 3)) ✠
[SS-4]
```

El sistema devuelve el objeto #4 que es un conjunto simplicial (marcado con el símbolo `SS`, por *Simplicial Set*), es decir, una versión combinatoria de la esfera S^3, y este objeto es asignado al símbolo `s3`. El siguiente paso es construir el complejo de cadenas canónicamente asociado a ese conjunto simplicial, paso intermedio para calcular la homología de S^3.

```
> (setf ccs3 (ss-cc s3)) ✠
[CC-7]
```

El sistema devuelve el objeto #7 que es un complejo de cadenas (`CC`, por *Chain Complex*), objeto que es asignado al símbolo `ccs3`. Finalmente, para calcular el tercer grupo de homología de la esfera introducimos

```
> (cc-homology ccs3 3) ✠
Homology in dimension 3:
Component Z
--- done ---
```

y el sistema devuelve la descripción del grupo de homología que, como es bien conocido, corresponde a $\mathbb{Z}$.

En el ejemplo anterior hay una cuestión fundamental que es la naturaleza finita de la versión combinatoria del espacio topológico de partida. Un ejemplo más interesante es plantearnos el cálculo de la homología de un espacio de naturaleza infinita. Vamos a suponer que queremos calcular el grupo de homología $H_5(\Omega^2 S^3)$, es decir, el quinto grupo de homología del segundo espacio de lazos de la esfera S^3. (El *espacio de lazos de un espacio* es el conjunto de funciones continuas de la circunferencia en el espacio que preservan el punto base, dotado de la topología compacto-abierta.) EAT construye una versión combinatoria del segundo espacio de lazos de la esfera S^3 codificado funcionalmente como un conjunto de funciones, versión combinatoria de naturaleza infinita que viene dada por un objeto conjunto simplicial.

```
> (setf l2s3 (loop-space (loop-space s3))) ✠
[SS-6]
```

Construimos el complejo de cadenas canónicamente asociado.

```
> (setf ccl2s3 (ss-cc l2s3)) ✠
[CC-8]
```

Sin embargo, al intentar calcular su homología por el mismo proceso que en el caso de la esfera:

*Las siguientes sentencias se han ejecutado en el sistema EAT bajo `Allegro Common Lisp`. El símbolo `>` representa al prompt Lisp y la cruz de malta ✠ corresponde a la tecla `<Return>`, que da paso a la evaluación de la sentencia.

```
> (cc-homology ccl2s3 5) ✠
;;Error:  CC-MAT cannot work with a LOCALLY-EFFECTIVE chain complex
```

se produce un error, no es posible.

Vamos a analizar las estructuras de datos anteriores viendo cómo son almacenadas en la memoria del ordenador para entender la diferencia esencial entre ambas, diferencia que procede de la distinta naturaleza, finita e infinita, de los espacios que codifican. Para ello utilizamos la función Common Lisp `inspect` (véase [116], página 698).

```
> (inspect s3) ✠
```

La anterior sentencia Lisp muestra la ventana de la figura 1.1.

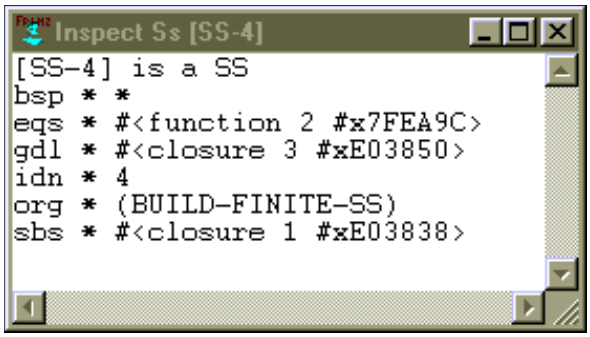


Figura 1.1: `inspect` de `s3`

La figura 1.1 es un ejemplo de cómo `inspect` muestra un objeto Lisp: una línea para cada campo del objeto apareciendo el nombre del campo delante del símbolo `*`. Vamos a explicar los campos del objeto conjunto simplicial anterior: `idn` y `org` podemos ignorarlos ya que contienen información útil desde el punto de vista de la ingeniería del software pero irrelevante desde el punto de vista de la especificación; en el campo `bsp` se codifica el punto base; `eqs` codifica una relación de equivalencia que define una igualdad entre símplices (más exactamente, entre sus representaciones); `gdl` codifica los operadores cara; finalmente, `sbs` codifica una función que asocia a cada número natural n una lista de símplices de dimensión n. Los campos más importantes de la estructura anterior (`eqs`, `gdl` y `sbs`) son de naturaleza funcional, pudiendo ser funciones primitivas (`#<function ... >`) o cierres léxicos definidos por el usuario (`#<closure ... >`). Esta característica se reflejará en las construcciones realizadas en esta memoria.

Una diferencia esencial entre las estructuras `s3` y `l2s3` es que la primera de ellas es una estructura finita, *efectiva* en terminología de Sergeraert [101], es una implementación directa de un conjunto simplicial finito, concretamente es una versión combinatoria del espacio topológico S^3. Sin embargo, `l2s3` es una implementación de un espacio infinito, implementación que recoge la no finitud del espacio y que contiene toda la información necesaria para realizar los cálculos. Así, podemos comparar símplices, aplicar los operadores cara, etc. Por ejemplo,

```
> (gdl l2s3 0 3 (asm nil (loop2 (asm nil (loop3 nil '<s3> 1)) 1))) ✠
<ASM 1-0 <<LOOP *>>>
```

construye la cara δ_0^3 de un símplice de `l2s3`. Evidentemente, en el caso de los objetos de naturaleza finita, se pueden realizar los cálculos del mismo modo. La siguiente sentencia aplica el operador cara δ_0^3 a un símplice de `s3`.

```
> (gdl s3 0 3 '<S3>) ✠
<ASM 1-0 *>
```

Vamos a aprovechar la evaluación anterior para comentar que en el operador podemos distinguir dos tipos de argumentos: el primero y el resto. El primer argumento determina el espacio ambiente en el que se realizan los cálculos. Como datos, son más bien "ocultos" ya que aunque su estructura pueda explorarse explícitamente por medio de `inspect` (ver figuras 1.1 y 1.2), su composición interna no puede mostrarse: los valores de los campos son cierres léxicos. La conclusión de esto es que los objetos asociados a `s3` y `l2s3` sólo tienen *comportamiento*, son *puramente observacionales* (una vez dados los argumentos para los cierres léxicos que son sus componentes). Por el contrario, el resto de argumentos son "visibles", en el sentido de que su estructura es completamente conocida. Esta característica la poseen también los resultados obtenidos al aplicar las operaciones. Por ejemplo, el símplice abstracto `<ASM 1-0 *>` es la degeneración $\eta_1\eta_0$ del punto base de la esfera denotado por `*`. Así, su estructura es completamente "visible" a partir de su representación en la máquina. Además, los datos "visibles" pueden ser de dos tipos: constantes, como por ejemplo `0` y `3`, o generados como `<ASM 1-0 *>`.

La diferencia esencial entre los objetos `s3` y `l2s3` se encuentra en el campo `sbs` y está relacionada con su distinta naturaleza, finita e infinita, respectivamente. Mostramos a continuación la estructura `l2s3` para compararlas.

```
> (inspect l2s3) ✠
```

La ventana `inspect` corresponde con la de la figura 1.2.

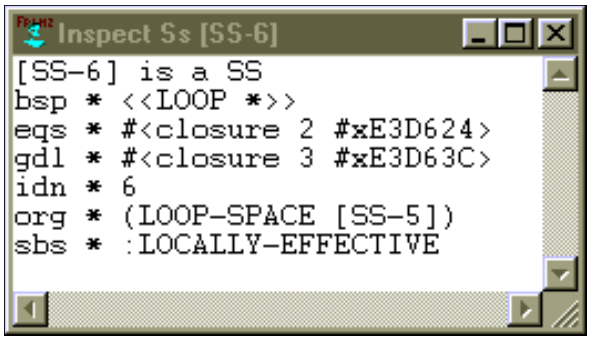


Figura 1.2: `inspect` de `l2s3`

En el caso del objeto `s3`, el campo `sbs` almacena una función que asocia a cada número natural n la lista de símplices no degenerados en esa dimensión, que es una lista finita. Sin embargo, en el caso del objeto `l2s3` en el campo `sbs` se almacena el símbolo `:LOCALLY-EFFECTIVE`. Esto es debido a que el conjunto simplicial que modela combinatorialmente el segundo espacio de lazos de la esfera S^3 es de naturaleza infinita (número infinito de símplices geométricos en alguna dimensión). (También podemos encontrarnos esta situación en el caso de un objeto de naturaleza finita cuya implementación en el sistema no recoja ninguna información acerca de su cardinalidad, por ejemplo, porque en el proceso de codificación se haya perdido esa información.) Esta diferencia entre los objetos `s3` y `l2s3` refleja la característica de efectividad o no de un objeto, es decir, la diferencia entre un objeto *efectivo* y un objeto *localmente efectivo* en terminología de Sergeraert. El objeto `l2s3` es un objeto *localmente efectivo*, es una implementación de un espacio infinito. En el caso de objetos que implementan espacios de naturaleza infinita, toda la información disponible es de naturaleza *local*, es decir, está relacionada con un

único elemento o con un conjunto finito de elementos que se encuentran próximos (en algún sentido geométrico o algebraico). Así, en el caso del objeto efectivo `s3` podemos obtener la lista de símplices no degenerados, por ejemplo en dimensión 3.

```
> (sbs s3 3) ✠
(<S3>)
```

La lista está formada por un único símplice, el símplice fundamental de S^3, llamado también `S3`. Sin embargo, en el caso del objeto `l2s3` no podemos obtener información similar ya que la "lista" de símplices no degenerados en dimensión 3 del espacio codificado es infinita

```
> (sbs l2s3 3) ✠
;; Error:  The simplicial set [SS-6] is locally-effective
```

o, visto de otra forma, no es información de naturaleza local, sino global a todo el conjunto simplicial.

El objeto `s3` es un objeto efectivo mientras que el objeto `l2s3` es localmente efectivo. Esa diferencia entre los objetos hace que no podamos calcular directamente la homología de `l2s3` tal y como hemos calculado la de `s3`, puesto que la homología de un espacio es información de carácter global. Pese a ello, por medio de los objetos localmente efectivos se resuelve el problema de representación en la máquina de espacios infinitos.

El siguiente problema del que nos ocupamos, una vez que hemos visto que en EAT se pueden representar espacios infinitos y hacer cálculos locales dentro de ellos, es el del cálculo de la homología de estos espacios, puesto que como ya hemos comentado es habitual que ésta sea de tipo finito y por tanto susceptible de ser calculada computacionalmente. Como herramienta para calcular la homología de espacios de naturaleza infinita Sergeraert en [112] introdujo la noción de *objeto con homología efectiva*. La idea es enriquecer los objetos con información adicional que permita aplicar sobre ellos un algoritmo de cálculo de su homología, aún en el caso de que los objetos sean de naturaleza infinita. Esta idea se apoya en que al emplear los conjuntos simpliciales como modelos combinatoriales de los espacios topológicos, el algoritmo de Veblen para el cálculo de la homología (introducido en [120]) se aplica sobre conjuntos simpliciales de naturaleza finita pero no sobre otros que, pese a no ser de naturaleza finita, tienen homología de tipo finito.

Volvemos a nuestro objetivo de calcular el quinto grupo de homología del segundo espacio de lazos de la esfera S^3 mediante EAT. A partir de su modelo combinatorio `s3` construimos el objeto con homología efectiva asociado a ese espacio.

```
> (setf s3eh (ess-sseh s3)) ✠
[SS-EH 0]
```

Seguidamente construimos el objeto con homología efectiva asociado al espacio $\Omega^2 S^3$.

```
> (setf l2s3eh (loop-space-eh (loop-space-eh s3eh))) ✠
[SS-EH 8]
```

El cálculo de la homología en dimensión 5 es obtenido por la siguiente evaluación

```
> (homology l2s3eh 5) ✠
Component Z/3Z
Component Z/2Z
---done---
```

siendo el resultado devuelto por el sistema $H_5(\Omega^2 S^3) = \mathbb{Z}_2 \oplus \mathbb{Z}_3$.

Un objeto con homología efectiva es un objeto "mixto" que presenta características efectivas y localmente efectivas, permitiendo aprovechar las ventajas que proporcionan ambas: la codificación localmente efectiva de un objeto de naturaleza infinita permite resolver las dificultades que deriven de la no finitud, mientras que la codificación efectiva permite obtener información de naturaleza global como es la homología del objeto. Los dos tipos de codificación están relacionados por medio de equivalencias de homotopía. Sin entrar en detalles, la idea es que las equivalencias de homotopía son una herramienta que permite calcular la homología de un objeto localmente efectivo ligándolo a un objeto efectivo con la misma homología.

En nuestro ejemplo, el objeto con homología efectiva asociado al espacio $\Omega^2 S^3$ tiene la estructura mostrada en la figura 1.3:

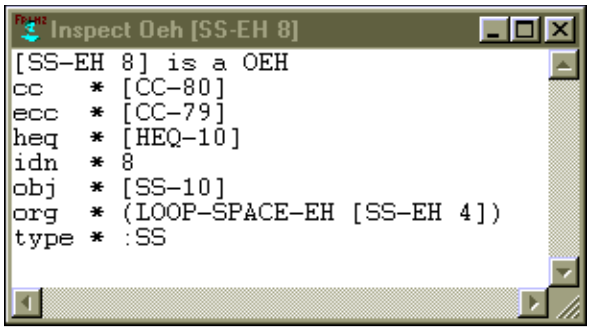


Figura 1.3: `inspect` de `l2s3eh`

Comentamos a continuación la información recogida en los campos de la estructura anterior: la información del campo `org` es irrelevante para nuestros propósitos; el campo `obj` recoge el objeto cuya homología se quiere calcular, en este caso una versión combinatorial del segundo espacio de lazos de la esfera S^3; `type` es el tipo del objeto recogido en el campo `obj`, conjunto simplicial como modelo combinatorial del objeto en este caso; el campo `cc` almacena el complejo de cadenas canónicamente asociado al objeto; el campo `ecc` recoge un complejo de cadenas efectivo; y, por último, el campo `heq` representa la equivalencia de homotopía entre los complejos de cadenas recogidos en los campos `cc` y `ecc`. Además si el objeto recogido en el campo `obj` es localmente efectivo su complejo de cadenas asociado (recogido en el campo `cc`) también lo es. Sin embargo, el complejo de cadenas recogido en el campo `ecc` siempre es efectivo y con la misma homología que el complejo de cadenas recogido en `cc`, puesto que son homotópicamente equivalentes por medio de la equivalencia de homotopía recogida en el campo `heq`. Si intentamos calcular el quinto grupo de homología del objeto `[CC-80]` (objeto recogido en el campo `cc`)

```
> (cc-homology (oeh-cc l2s3eh) 5) ✠
;; Error:  CC-MAT cannot work with a LOCALLY-EFFECTIVE chain complex.
```

el sistema falla ya que no tiene información suficiente para calcular la homología del complejo de

cadenas [CC-80] que es localmente efectivo, como puede verse en la figura 1.4 correspondiente a la ventana mostrada al evaluar

```
> (inspect (CC 80)) ✠
```

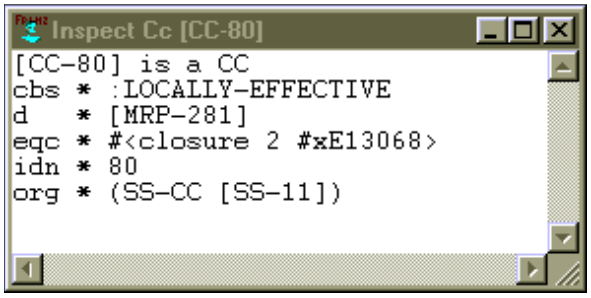


Figura 1.4: `inspect` de [CC-80]

Los complejos de cadenas son uno de los objetos más básicos en EAT. Sus principales campos son: `eqc` que codifica la igualdad entre elementos; `d` que codifica la diferencial; y, `cbs` que en el caso efectivo codifica una función que devuelve para cada entero p la lista de generadores del grupo de grado p, y el símbolo `:LOCALLY-EFFECTIVE` en el caso de un complejo de cadenas localmente efectivo.

Sin embargo, el complejo de cadenas [CC-79] es efectivo, como muestra la figura 1.5 obtenida por evaluación de

```
> (inspect (CC 79)) ✠
```

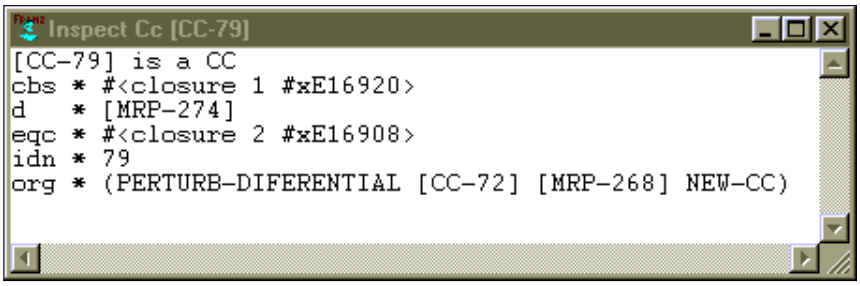


Figura 1.5: `inspect` de [CC-79]

En este caso podemos calcular su quinto grupo de homología por evaluación de la siguiente sentencia.

```
> (cc-homology (oeh-ecc l2s3eh) 5) ✠
Component Z/3Z
Component Z/2Z
---done---
```

El sistema devuelve el quinto grupo de homología del segundo espacio de lazos de la esfera S^3.

Como observación, notar que a partir de un objeto efectivo también puede construirse un objeto con homología efectiva. Como todo objeto efectivo es, en particular, localmente efectivo, en este caso el complejo de cadenas asociado al objeto y recogido en el campo `cc` será efectivo.

Incluimos a continuación un ejemplo más sofisticado de cálculo realizado por EAT. Vamos a calcular el quinto grupo de homología del espacio $\Omega(\Omega S^3 \cup_2 D^3)$, espacio de lazos del espacio obtenido pegando el disco D^3 al espacio de lazos de la esfera S^3 a través de una aplicación de grado 2. Para ello, construimos en primer lugar el objeto con homología efectiva asociado al espacio de lazos de la esfera S^3

```
> (setf l1s3eh (loop-space-eh s3eh)) ✠
[SS-EH 8]
```

objeto asignado al símbolo `l1s3eh`. A continuación construimos otro objeto, con homología efectiva, asociado al espacio $\Omega S^3 \cup_2 D^3$.

```
> (setf dl1s3eh (disk-paste-eh l1s3 3 (list (asm nil (loop3 nil '<s3> 1))
(null-asm-loop 2) (asm nil (loop3 nil '<s3> 1)) (null-asm-loop 2)) :new
'<D3>)) ✠
[SS-EH 9]
```

El último objeto que necesitamos construir es el objeto con homología efectiva asociado al espacio de lazos del objeto anterior.

```
> (setf l1dl1s3eh (loop-space-eh dl1s3eh)) ✠
[SS-EH 13]
```

El objeto asociado al símbolo `l1dl1s3eh` es un objeto con homología efectiva a partir del que calculamos el quinto grupo de homología del espacio de partida, mediante la siguiente sentencia

```
> (homology l1dl1s3eh 5) ✠
Component Z/2Z
Component Z/2Z
Component Z/2Z
Component Z/2Z
Component Z/2Z
Component Z/2Z
Component Z
---done---
```

y el resultado $H_5\Omega(\Omega S^3 \cup_2 D^3) = (\mathbb{Z}_2)^6 \oplus \mathbb{Z}$ es obtenido en 15 segundos en un PC133MHZ. La cuestión a plantearse es la fiabilidad del resultado obtenido, ya que no parece muy razonable confirmarlo ni refutarlo directamente. Por este motivo, consideramos importante el análisis de la corrección del sistema, tal y como ya hemos comentado en la introducción.

1.2 Conjuntos, Categorías y Especificación Algebraica

1.2.1 Conjuntos

El objetivo de esta sección es fijar las definiciones, terminología y notación relativa a la Teoría de Conjuntos, que vamos a utilizar en este trabajo. Es muy extensa la literatura sobre Teoría de Conjuntos, aquí hemos utilizado [87] como referencia general y [72] como más específica para algunas definiciones concretas.

Pese a que todo lo que incluimos en esta sección son conocimientos básicos en Matemáticas, preferimos dar un repaso explícito de ellos, puesto que algunas de las definiciones que incluimos a continuación son fundamentales en determinadas partes de este trabajo.

Como primitiva consideramos la noción de conjunto. Entendemos que, por definición, cada conjunto está dotado de una igualdad entre sus elementos. Usaremos los conceptos de conjunto vacío ($\emptyset$), subconjunto ($\subseteq$), conjunto diferencia ($\setminus$), el conjunto de partes de un conjunto ($\wp(-)$), producto cartesiano ($\times$) y unión e intersección ($\cup$, $\cap$, respectivamente). Si X es un conjunto, como es habitual X^* denota al conjunto formado por todas las secuencias de elementos de X, incluyendo la secuencia vacía, que representaremos por []. Como es habitual, denotaremos por $\mathbb{N}$ al conjunto de los números naturales, por $\mathbb{Z}$ al de los números enteros y por $\mathbb{R}$ al de los números reales.

Si J es un conjunto y $\{A_j\}_{j\in J}$ es una familia indexada por J de conjuntos, diremos que la familia anterior es un *J-conjunto*. Todas las nociones consideradas anteriormente sobre conjuntos se generalizan de forma natural a familias indexadas de conjuntos.

Es bien conocido que la Teoría de Conjuntos presenta problemas en sus Fundamentos, problemas que provienen del "tamaño" del marco en el que se está trabajando. En cualquier área concreta de las Matemáticas lo que se busca es un marco de referencia seguro en el que trabajar. Para ello, un buen punto de partida es establecer un *Universo*. La idea subyacente en la definición de un universo es que se puedan realizar dentro de él las operaciones elementales que se realizan con conjuntos, es decir, que las operaciones elementales sean internas al propio universo. En ese sentido, un universo proporciona un marco de referencia en el que trabajar. En [16] podemos encontrar una definición formal de universo que correponde con las ideas que acabamos de expresar informalmente y que no recogemos explícitamente.

La forma habitual de trabajar en Matemáticas es fijar un universo y considerar conjuntos en él. Por ello, en lo que sigue supondremos que siempre se dispone implícita o explícitamente de un universo fijado. Un ejemplo de universo es el formado por todos los objetos Lisp, en el que nos situaremos en las partes de este trabajo más cercanas a la programación real y al que nos referiremos por universo Lisp.

Va a ser fundamental en este trabajo la noción de relación de equivalencia.

1.2.1.1 Relaciones

Definición 1.2.1 *Sean A y B conjuntos. Una* relación entre A y B *es un subconjunto $R \subseteq A \times B$.*

Si $A = B$ entonces hablamos de relación (binaria) en A.

Recogemos a continuación algunas propiedades que pueden verificar las relaciones:

Definición 1.2.2 *Sea A un conjunto y R una relación en A.*

- *Se dice que R es* reflexiva *si $(a, a) \in R$, para todo $a \in A$.*
- *Se dice que R es* simétrica *si para todo $a_1, a_2 \in A$, $(a_1, a_2) \in R$ implica que $(a_2, a_1) \in R$.*
- *Se dice que R es* antisimétrica *si para todo $a_1, a_2 \in A$, $(a_1, a_2) \in R$ y $(a_2, a_1) \in R$ implican que $a_1 = a_2$.*
- *Se dice que R es* transitiva *si para todo $a_1, a_2, a_3 \in A$, $(a_1, a_2) \in R$ y $(a_2, a_3) \in R$ implican que $(a_1, a_3) \in R$.*
- *La relación R se dice de* equivalencia *si es reflexiva, simétrica y transitiva. Si a y b son elementos de A y R es una relación de equivalencia en A, se dice que a y b son* elementos equivalentes en A módulo R *si $(a, b) \in R$.*
- *Se dice que R es una* relación de orden parcial *si es reflexiva, antisimétrica y transitiva.*

Definición 1.2.3 *Sea A un conjunto y R una relación de equivalencia en A. Para cada elemento x de A,* la clase de equivalencia de x módulo R *es el conjunto de todos los elementos de A equivalentes a x.*

Denotamos por $[x]$ a la clase de equivalencia de un elemento x módulo R.

Definición 1.2.4 *Si R es una relación de equivalencia en un conjunto A, el conjunto de todas las clases de equivalencia módulo R se denomina* conjunto cociente de A por R *y se denota por A/R.*

Definición 1.2.5 *Si R_1 y R_2 son relaciones de equivalencia en un conjunto A, se dice que R_1 es* más fina *o* más pequeña *que R_2 si $R_1 \subseteq R_2$, es decir, si para cada par $a, a' \in A$, $(a, a') \in R_1$ implica que $(a, a') \in R_2$.*

En este caso se dice que R_2 es menos fina *o* más grande *que R_1.*

Informalmente, una relación es más pequeña que otra si relaciona menos elementos, por lo que las clases de equivalencia tienen menos elementos, y el conjunto cociente posee más clases de equivalencia.

Definición 1.2.6 *Si R_1 y R_2 son relaciones de equivalencia en un conjunto A se dice que R_2* es compatible con R_1 *si R_2 es más grande que R_1.*

Si $R \subseteq A \times A$ es de equivalencia y B es un subconjunto de A, la restricción $R_B = R \cap (B \times B)$ es una relación de equivalencia en B, que denominamos *relación de equivalencia heredada de R.*

Las relaciones de equivalencia identifican elementos dentro de un conjunto por lo que también se les denomina *igualdades*. Al trabajar en un conjunto en el que hay definida una relación de equivalencia (nos referimos por trabajar a definir una función que sale de él, tomar un subconjunto, etc.) es necesario tener en cuenta que las operaciones que estemos realizando sean compatibles con la relación, esto es, pueden ser definidas en el nivel del conjunto cociente. En Matemáticas, cuando en un conjunto hay definida una relación de equivalencia lo natural es realizar el paso al conjunto cociente. Sin embargo, para definir correctamente algunas de las

nociones que daremos más adelante en este trabajo resulta más conveniente la idea de conjunto con igualdad explícita.

Llamaremos *conjunto con igualdad* a un par $(A, =_R)$ donde A es un conjunto y R es la igualdad definida en A por una relación de equivalencia R. (Si a_1 y a_2 son elementos de A, entonces a_1 es *igual* a a_2 en $(A, =_R)$ si y solo si $(a_1, a_2) \in R$.)

Definición 1.2.7 *Sea $(A, =_{R_A})$ un conjunto con igualdad, B un subconjunto de A y R_B la relación de equivalencia en B heredada de la de A. Si $\tilde{R}_B$ es una relación de equivalencia en B, el conjunto con igualdad $(B, =_{\tilde{R}_B})$ es* subconjunto de $(A, =_{R_A})$ *si la relación de equivalencia $\tilde{R}_B$ es más grande que la relación R_B.*

Si $(A, =_{R_A})$ es un conjunto con igualdad, cualquier subconjunto $(B, =_{R_B})$ de $(A, =_{R_A})$ con igualdad la heredada de $=_{R_A}$ lo denotaremos simplemente, si no hay lugar a confusión, por B.

1.2.1.2 Funciones

Definición 1.2.8 *Sean A y B conjuntos. Una* función parcial *o* aplicación parcial de A en B *es una relación $f \subseteq A \times B$ verificando que, para cada $a \in A$ existe, como mucho, un $b \in B$ tal que $(a, b) \in f$.*

Escribiremos $f(a) = b$ (o $f(a) := b$) en lugar de $f(a, b)$ y diremos que $f(a)$ es indefinido si no existe $b \in B$ tal que $(a, b) \in f$. En ambos casos, nos referiremos a $f(a)$ como el *valor* de la función f en el *argumento* a.

Denotaremos por $f\colon A \to B$ a una función parcial f de A en B. El conjunto A se llama *dominio de f* y el conjunto B *codominio de f*. Se llama *dominio de definición de f* al subconjunto de A dado por $Def(f) = \{a \in A \mid \text{existe } b \in B \text{ tal que } (a, b) \in f\}$. Al conjunto $\{b \in B \mid \text{existe } a \in A \text{ con } (a, b) \in f\}$ se le denomina *conjunto imagen de f* y se suele denotar por $Im\ f$.

Una función parcial $f\colon A \to B$ solo queda completamente determinada por la cuaterna $(f, Def(f), A, B)$. Así, para referirnos a la función parcial anterior emplearemos la notación $(f, Def(f), A, B)$ o bien $(f\colon A \to B, Def(f))$. Si el dominio de una función parcial $f\colon A \to B$ es un producto $A_1 \times \cdots \times A_n$, para algún $n \in \mathbb{N}$, entonces denotaremos a la función parcial por $(f, Def(f), A_1, \ldots, A_n, B)$.

Si $Def(f) = A$ hablaremos de *función total*. Por tanto, toda función total es, en particular, función parcial.

Cuando en esta memoria hablemos simplemente de función será porque el contexto aclara si nos referimos a *función parcial* o a *función total* o porque no es relevante el hecho de que las funciones sean totales o parciales.

Definición 1.2.9 *Sean $f\colon A \to B$ y $g\colon B \to C$ funciones parciales. La* composición de f y g *es la función $g \circ f\colon A \to C$ definida por $(g \circ f)(a) = g(f(a))$, para todo $a \in Def(f)$ tal que $f(a) \in Def(g)$.*

Las funciones pueden verificar determinadas propiedades, algunas de las cuales recogemos a continuación.

Definición 1.2.10 *Sea $f\colon A \to B$ una función parcial.*

- f *es* inyectiva *si para cada* $b \in B$*, existe a lo más un elemento* $a \in A$ *tal que* $f(a) = b$.
- f *es* sobreyectiva *si para cada* $b \in B$*, existe al menos un elemento* $a \in A$ *tal que* $f(a) = b$.
- f *es* biyectiva *si es total, inyectiva y sobreyectiva. En ese caso diremos que los conjuntos* A *y* B *son* biyectivos *y lo representaremos por* $A \cong B$.

  Si $f\colon A \to B$ *es una biyección, entonces existe otra función total* $f^{-1}\colon B \to A$ *verificando que* $f^{-1} \circ f = 1_A$ *y* $f \circ f^{-1} = 1_B$*. La función* f^{-1} *se dice* inversa de la función f.

Nos interesa señalar la siguiente terminología y notación relacionada con funciones.

- Si A es un conjunto, la función *identidad sobre* A, $1_A\colon A \to A$, es la función definida por $1_A(a) := a$, para todo $a \in A$.
- Si A y B son conjuntos con $A \subseteq B$, la función *inclusión de* A *en* B, $i_A\colon A \to B$, es la función tal que $i_A(a) := a$, para todo $a \in A$.
- Sea $f\colon A \to B$ una función parcial y $C \subseteq A$. La *restricción de* f *a* C es la función parcial $f|_C\colon C \to B$ con dominio de definición $Def(f|_C) = Def(f) \cap C$ y definida por $f|_C(c) := f(c)$, para todo $c \in Def(f|_C)$.

  Si una función es total su restricción a cualquier subconjunto del dominio es otra función total. Si la función es parcial su restricción puede ser total o parcial.
- Si A es un subconjunto de un conjunto B, se denomina *función característica del conjunto* A, y se denota por $\chi_A\colon B \to \{true, false\}$, a la función total definida por $\chi_A(b) := true$ para todo $b \in A$, y $\chi_A(b) := false$ para todo $b \notin A$.

Si X e Y son conjuntos, denotamos por Y^X al conjunto de las funciones parciales de X en Y. Si estamos en un marco en el que todas las funciones son totales, utilizaremos la misma notación, Y^X, para el conjunto de funciones totales de X en Y.

Las funciones con codominio el conjunto de valores booleanos, $\{true, false\}$, se denominan *predicados.*

Definición 1.2.11 *Sea* $(f\colon A \to B, A')$ *una función parcial y sean* $=_A$*,* $=_{A'}$ *e* $=_B$ *relaciones de equivalencia en* A*,* A' *y* B*, respectivamente, de forma que* $(A', =_{A'})$ *es subconjunto de* $(A, =_A)$*. Se dice que* f respeta las igualdades *si para todo par* a_1, a_2 *de elementos de* A' *tales que* $a_1 =_{A'} a_2$ *se tiene que* $f(a_1) =_B f(a_2)$.

En este caso podemos ver la función f *como* $\big(f\colon (A, =_A) \to (B, =_B), (A', =_{A'})\big)$.

Toda función parcial $(f\colon A \to B, Def(f))$ define una relación de equivalencia en su dominio de definición, relación de equivalencia que denotamos por $=_f$ y que viene definida del siguiente modo: si $a_1, a_2 \in Def(f)$, $a_1 =_f a_2$ si y solo si $f(a_1) = f(a_2)$. A la relación $=_f$ la denominamos *equivalencia definida por la función* f. Si en el codominio B hay definida una igualdad explícita $=_B$, la relación de equivalencia $=_f$ viene definida del siguiente modo: para todo par $a_1, a_2 \in Def(f)$, $a_1 =_f a_2$ si y solo si $f(a_1) =_B f(a_2)$.

Definición 1.2.12 *Sean* $(A, =_A)$ *y* $(B, =_B)$ *conjuntos con igualdad. Una función* $(f\colon A \to B, Def(f))$ *es* compatible con las igualdades *si la restricción de* $=_A$ *a* $Def(f)$ *es más fina que* $=_f$.

En este caso, $(f\colon (A,=_A) \to (B,=_B), Def(f))$ *también es función parcial (*$Def(f)$ *con la igualdad heredada como subconjunto de* $(A,=_A)$*).*

Usaremos también la noción de aplicación entre familias de conjuntos. Si J es un conjunto y $A = \{A_j\}_{j\in J}$ y $B = \{B_j\}_{j\in J}$ son J-conjuntos, $(f\colon A \to B, Def(f))$ denota a la familia de funciones $f = \{f_j\colon A_j \to B_j, Def(f_j)\}_{j\in J}$. Se dice que una familia indexada de funciones posee una propiedad si todas las funciones de la familia la poseen.

1.2.2 Categorías

La Teoría de Categorías se caracteriza por su alto nivel de abstracción y proporciona un lenguaje útil para unificar conceptos en distintas áreas de Matemáticas. Es muy extensa la literatura general sobre Teoría de Categorías, entre la que señalamos los libros [16], [17], [74], [79]. Pero además, son numerosos los trabajos que en diferentes áreas de Matemáticas se apoyan en Teoría de Categorías y en muchos de los cuales también pueden encontrarse sus nociones básicas. Ejemplos de este tipo de referencias son [10], [107] y [37], en los que se desarrollan nociones de Teoría de Categorías que son útiles, en este caso concreto, en el área de las Ciencias de la Computación.

No pretendemos ser en absoluto exhaustivos. Nos vamos a limitar a recordar algunas nociones sobre categorías que son necesarias para seguir el presente trabajo. Comenzamos por la propia noción de categoría. La que aquí damos corresponde con lo que en la literatura se denomina *categoría localmente pequeña.*

Definición 1.2.13 *Una* categoría $\mathcal{C}$ *consta de*

- *una clase* $Obj(\mathcal{C})$ *cuyos elementos se denominan* objetos de la categoría $\mathcal{C}$*;*
- *para cada par de objetos* $A, B \in Obj(\mathcal{C})$*, un conjunto* $Morf_{\mathcal{C}}(A,B)$ *cuyos elementos se denominan* morfismos de A en B*;*
- *para cada terna* A, B, C *de objetos de* $\mathcal{C}$*, una aplicación* $: Morf_{\mathcal{C}}(A,B) \times Morf_{\mathcal{C}}(B,C) \to Morf_{\mathcal{C}}(A,C)$ *que se denomina* composición*; la composición de un par* (f,g) *se denota por* $g \circ f$*;*
- *para cada* A *objeto de* $\mathcal{C}$ *un morfismo del conjunto* $Morf_{\mathcal{C}}(A,A)$*, denotado por* 1_A*, que se denomina* identidad sobre A*;*

de forma que se verifican los siguientes axiomas:

Ax1. la composición es asociativa: dados f *morfismo de* $Morf_{\mathcal{C}}(A,B)$*,* g *morfismo de* $Morf_{\mathcal{C}}(B,C)$ *y* h *morfismo de* $Morf_{\mathcal{C}}(C,D)$*, se verifica la siguiente igualdad*

$$h \circ (g \circ f) = (h \circ g) \circ f$$

Ax2. para todo f *morfismo de* $Morf_{\mathcal{C}}(A,B)$ *y para todo* g *morfismo de* $Morf_{\mathcal{C}}(B,C)$*, se tienen las siguientes igualdades:*

$$1_B \circ f = f;\ g \circ 1_B = g$$

Escribiremos $f\colon A \to B$ para representar $f \in Morf_{\mathcal{C}}(A,B)$ y llamaremos al objeto A *dominio* y al objeto B *codominio* del morfismo f.

Si $\mathcal{C}$ es una categoría escribiremos $\mathcal{C} = \langle Obj(\mathcal{C}), Morf(\mathcal{C})\rangle$ siendo $Morf(\mathcal{C})$ la clase de todos los morfismos entre pares de objetos de $\mathcal{C}$.

Ejemplo 1.2.14

- Los conjuntos constituyen un ejemplo fundamental de categoría. Se define la *Categoría de Conjuntos*, denotada habitualmente por *Set*, como la categoría cuyos objetos son los conjuntos y cuyos morfismos son las aplicaciones (totales) entre conjuntos. La composición es la composición habitual de funciones y el morfismo identidad para cada objeto es la aplicación identidad.

- Definimos otra categoría que denotamos por $PSet$ con objetos los conjuntos y con morfismos las funciones parciales. La operación de composición es la composición de funciones parciales, tal y como la hemos definido anteriormente.

- Cada estructura matemática definida sobre un conjunto lleva asociada su correspondiente noción de morfismo: "aplicación que conserva la estructura". Cada una de ellas da lugar a una categoría que se obtiene como especialización de la categoría *Set*. Algunos ejemplos de este tipo de categorías son los siguientes:

  1. La *Categoría de grupos*, denotada por *Grp*, tiene por objetos a los grupos y como morfismos los homomorfismos de grupos.
  2. La *Categoría de grupos abelianos*, denotada por *GrpAb*, tiene como objetos a los grupos abelianos y los mismos morfismos que la categoría de grupos.
  3. *Anll* denota a la *Categoría de anillos*, cuyos objetos son los anillos y cuyos morfismos son los homomorfismos de anillos.

  Las categorías anteriores son todas ellas ejemplos de un tipo particular de categorías denominadas *categorías concretas*, que son aquellas en las que los objetos son conjuntos con estructura y los morfismos son aplicaciones conjuntistas que cumplen determinadas propiedades relacionadas con la estructura de los objetos.

- Un ejemplo de categoría diferente de los anteriores es la *Categoría de matrices*. Sus objetos son los números naturales, los morfismos entre dos objetos n y m son las matrices de orden $n \times m$, la composición es el producto de matrices y la identidad de cada objeto n la matriz identidad de orden n.

□

La siguiente definición recoge un tipo especial de categorías que aparecerán en nuestro trabajo.

Definición 1.2.15 *Una categoría se dice* discreta *si los únicos morfismos son las identidades.*

Una categoría discreta queda determinada únicamente por la clase de sus objetos. En este sentido, una categoría discreta puede identificarse con una clase.

La noción de isomorfismo en una categoría viene dada como sigue.

Definición 1.2.16 *Sea $\mathcal{C}$ una categoría y $f\colon A \to B$ un morfismo en $\mathcal{C}$. Se dice que f* es un isomorfismo en $\mathcal{C}$ entre A y B *si existe un morfismo en $\mathcal{C}$, $g\colon B \to A$, tal que $g \circ f = 1_A$ y $f \circ g = 1_B$.*

Definición 1.2.17 *Dos objetos A y B de una categoría $\mathcal{C}$ se dicen* isomorfos en $\mathcal{C}$ *si existe $f\colon A \to B$ isomorfismo en $\mathcal{C}$. Se denota por $A \cong B$.*

Ejemplo 1.2.18 En la categoría de conjuntos *Set* los isomorfismos son las aplicaciones biyectivas. En las categorías de grupos, grupos abelianos y anillos son los homomorfismos biyectivos. En la categoría de matrices son las matrices cuadradas invertibles. □

1.2.2.1 Funtores

A su vez, las categorías pueden ser consideradas como objetos de una categoría, la categoría de categorías. Damos a continuación la noción de funtor, es decir, de morfismo entre categorías.

Interpretando una categoría en su sentido más amplio, como marco matemático, un funtor entre categorías puede interpretarse como un cambio de marco de referencia. Desde el punto de vista de las categorías como estructuras matemáticas, los funtores son aplicaciones entre categorías que preservan la estructura de categoría. A continuación, damos su definición formal.

Definición 1.2.19 *Sean $\mathcal{C}$ y $\mathcal{D}$ categorías. Un* funtor F de $\mathcal{C}$ en $\mathcal{D}$, *denotado por $F\colon \mathcal{C} \to \mathcal{D}$, asigna:*

- *a cada objeto A de $\mathcal{C}$, un objeto $F(A)$ de $\mathcal{D}$;*
- *a cada morfismo $f\colon A \to B$ un morfismo $F(f)\colon F(A) \to F(B)$*

de forma compatible con la estructura de categoría, es decir, verificando las siguientes condiciones:

i) para cada par de morfismos $f \in Morf_{\mathcal{C}}(A, B)$, $g \in Morf_{\mathcal{C}}(B, C)$, se tiene que $F(g \circ f) = F(g) \circ F(f)$;

ii) para cada objeto A de $\mathcal{C}$, $F(1_A) = 1_{F(A)}$.

Hay un modo natural de definir una composición de funtores que resulta ser asociativa y, para cada categoría, existe el correspondiente funtor identidad. Sin embargo, las categorías junto con los funtores no constituyen una categoría tal y como nosotros las hemos definido. Hemos exigido que la clase de morfismos entre dos objetos de una categoría sea un conjunto, algo que no podemos asegurar al considerar como objetos las categorías y como morfismos los funtores entre ellas. Es decir, la "categoría de categorías" no es una categoría localmente pequeña.

Definición 1.2.20 *Una categoría $\mathcal{C}$ se dice* pequeña *si la clase de sus objetos es un conjunto.*

En nuestro caso, la clase de los morfismos de una categoría pequeña también será un conjunto. Las categorías pequeñas junto con los funtores entre ellas constituyen una categoría que denotamos por Cat.

Un funtor $F\colon \mathcal{C} \to \mathcal{D}$ entre categorías pequeñas podemos verlo como un par de aplicaciones $(F_{Obj}\colon Obj(\mathcal{C}) \to Obj(\mathcal{D}),\ F_{Morf}\colon Morf(\mathcal{C}) \to Morf(\mathcal{D}))$.

A partir de ahora todas las categorías que aparezcan suponemos que son categorías pequeñas, teniendo en cuenta que muchas de las definiciones y de los resultados recogidos a continuación son válidos para categorías generales.

Ejemplo 1.2.21

1. Sean $\mathcal{C}$ y $\mathcal{D}$ dos categorías y sea D un objeto de $\mathcal{D}$. El *funtor constante a* D es el funtor $\Delta_D\colon \mathcal{C} \to \mathcal{D}$ que lleva todos los objetos de $\mathcal{C}$ sobre el objeto D y todos los morfismos de $\mathcal{C}$ sobre la identidad en $\mathcal{D}$ del objeto D.

2. Podemos definir el funtor inclusión $i\colon Set \to PSet$, definido como la identidad sobre todos los objetos y morfismos de Set.

3. Otro tipo de ejemplos lo proporcionan los llamados *funtores olvido*. Los funtores olvido son funtores entre categorías concretas que olvidan parte de la estructura. Por ejemplo, quedándose únicamente con la parte conjuntista de la categoría inicial. Así, el funtor olvido $U_{Grp}\colon Grp \to Set$ lleva cada grupo al conjunto sobre el que está definido y cada homomorfismo de grupos f a la propia aplicación f.

□

Damos a continuación cierta terminología relacionada con funtores.

Definición 1.2.22 *Sea $F\colon \mathcal{C} \to \mathcal{D}$ un funtor entre categorías.*

- *F se dice* fiel *si para cada par de objetos A y B de $\mathcal{C}$, la aplicación $F\colon Morf_{\mathcal{C}}(A,B) \to Morf_{\mathcal{D}}(F(A),F(B))$ es inyectiva. Es decir, F es fiel si es inyectivo sobre los morfismos.*

- *F se dice* pleno *si para cada par de objetos A y B de $\mathcal{C}$, la aplicación $F\colon Morf_{\mathcal{C}}(A,B) \to Morf_{\mathcal{D}}(F(A),F(B))$ es sobreyectiva. Dicho de otro modo, F es pleno si es sobreyectivo sobre los morfismos.*

- *F es una* inmersión *si es fiel, pleno y verifica que para cada par de objetos A y B de $\mathcal{C}$, si $F(A) = F(B)$ entonces $A = B$ (es decir, es inyectivo sobre los objetos).*

- *F es* isomorfismo *si existe un funtor $G\colon \mathcal{D} \to \mathcal{C}$ tal que $G \circ F = 1_{\mathcal{C}}$ y $F \circ G = 1_{\mathcal{D}}$. En este caso se denota por $F \cong G$ y se dice que $\mathcal{C}$ y $\mathcal{D}$* son categorías isomorfas *(se denota $\mathcal{C} \cong \mathcal{D}$).*

La noción de isomorfismo entre categorías es muy fuerte, tanto que no identifica categorías que pueden considerarse esencialmente la misma. A continuación, damos otra noción, un poco más débil, que suele resultar más adecuada que la de categorías isomorfas.

Definición 1.2.23 *Un funtor $F\colon \mathcal{C} \to \mathcal{D}$ es una* equivalencia *si es fiel, pleno y verifica que para cada D objeto de $\mathcal{D}$, existe C, objeto de $\mathcal{C}$, tal que los objetos $F(C)$ y D son isomorfos en $\mathcal{D}$.*

Dos categorías se dicen *equivalentes* si existe una equivalencia entre ellas, se denota por $\mathcal{C} \simeq \mathcal{D}$.

La idea subyacente en la definición anterior, tal y como viene expresada en [10] (página 73), es que no solo cada objeto de $\mathcal{D}$ es isomorfo a uno de la imagen de la equivalencia, sino que además, los isomorfismos son compatibles con los morfismos de $\mathcal{D}$; y, análogamente para la categoría $\mathcal{C}$. Por ello, la noción de equivalencia es la que recoge realmente la idea de que dos categorías sean la misma.

1.2.2.2 Construcciones sobre categorías

Introducimos a continuación construcciones generales en categorías que usaremos a lo largo de la presente memoria.

Definición 1.2.24 *Si $\mathcal{C}$ es una categoría, una* subcategoría $\mathcal{C}_0$ de $\mathcal{C}$ *es una categoría formada por algunos de los objetos y por algunos de los morfismos de $\mathcal{C}$.*

Para cada subcategoría $\mathcal{C}_0$ de una categoría $\mathcal{C}$ se puede definir, de forma natural, el funtor *inclusión* $i\colon \mathcal{C}_0 \to \mathcal{C}$, funtor que siempre es fiel.

Definición 1.2.25 *Sea $\mathcal{C}$ una categoría y $\mathcal{C}_0$ una subcategoría suya. Se dice que $\mathcal{C}_0$ es* subcategoría plena de $\mathcal{C}$ *si el funtor inclusión es pleno.*

La idea de una subcategoría plena es que en el paso a la subcategoría no se hayan perdido ninguna de las relaciones entre los objetos de la categoría, es decir, el conjunto de morfismos entre dos objetos de la subcategoría coincide con el conjunto de morfismos entre esos dos objetos en la categoría.

Ejemplo 1.2.26 *Set* es una subcategoría no plena de *PSet*. *GrpAb* es subcategoría plena de *Grp*.

□

Definición 1.2.27 *Sea $\mathcal{C}$ una categoría y $\mathcal{C}_0$ una subcategoría de $\mathcal{C}$. Se dice que $\mathcal{C}_0$ es* cerrada por isomorfismos *si todos los objetos de $\mathcal{C}$ isomorfos a algún objeto de $\mathcal{C}_0$, están a su vez en $\mathcal{C}_0$.*

En la siguiente definición se explica la construcción de las categorías "slices", que se usará más adelante en el trabajo.

Definición 1.2.28 *Sean $\mathcal{C}$ una categoría y A un objeto de $\mathcal{C}$. La* categoría "slice" *$\mathcal{C}/A$ tiene por objetos los morfismos $f\colon B \to A$ de $\mathcal{C}$; para cada par de objetos $f\colon B \to A$ y $f'\colon B' \to A$ de $\mathcal{C}/A$, el conjunto de morfismos $Mor_{\mathcal{C}/A}(B, B')$ está formado por los morfismos $h\colon B \to B'$ de $\mathcal{C}$ tales que $f = f' \circ h$; la composición viene dada por la composición de $\mathcal{C}$; y la identidad de cada objeto $f\colon B \to A$ viene dada por la identidad 1_B de $\mathcal{C}$.*

Nuestro interés se centra en categorías "slices" sobre la categoría *Set* de conjuntos. Así, si A es un conjunto, cada objeto $f\colon B \to A$ de Set/A se puede interpretar como una familia de conjuntos indexada por A, la familia $\left\{f^{-1}(a)\right\}_{a\in A}$. Esto hace que podamos pensar en Set/A como la categoría de las familias de conjuntos indexadas por A con morfismos las familias

indexadas por A de aplicaciones entre conjuntos. Estas categorías aparecerán explícitamente en este trabajo.

Otras nociones categoriales que serán muy usadas en este trabajo son coproducto y objeto (familia) final.

Definición 1.2.29 *Sea J un conjunto y sea $\{A_j\}_{j\in J}$ una familia de objetos de una categoría $\mathcal{C}$. El* coproducto de la familia $\{A_j\}_{j\in J}$ *es un par $\left(A, \{i_j\}_{j\in J}\right)$ donde A es un objeto de $\mathcal{C}$ y, para cada $j \in J$, $i_j \colon A_j \to A$ es un morfismo de $\mathcal{C}$, verificando que, para cualquier otro objeto X de $\mathcal{C}$ y cualquier otra familia de morfismos $\{s_j \colon A_j \to X\}_{j\in J}$ de $\mathcal{C}$, existe un único morfismo $F \colon A \to X$ en $\mathcal{C}$ tal que, para cada $j \in J$ se tiene que $F \circ i_j = s_j$.*

Al coproducto de la familia de objetos $\{A_j\}_{j\in J}$ se le suele denotar por $\coprod_{j\in J} A_j$; a los morfismos de la familia $\{i_j\}_{j\in J}$ se les denomina *inclusiones canónicas*.

Ejemplo 1.2.30 En la categoría de conjuntos Set, el coproducto de una familia de objetos es la unión disjunta, de forma que si $\{C_j\}_{j\in J}$ es una familia de conjuntos, $\coprod_{j\in J} C_j = \left\{(x, j) \mid j \in J \text{ y } x \in C_j\right\}$, siendo las aplicaciones canónicas $i_j \colon C_j \to \coprod_{j\in J} C_j$, para cada $j \in J$, las inclusiones, $i_j(x) := (x, j)$.

En la categoría de grupos, el coproducto de una familia de grupos es, lo que en Teoría de Grupos se denomina habitualmente, el producto libre de grupos (véase [16], página 46, para más detalles).

En la categoría de grupos abelianos, el coproducto coincide con la suma directa.

□

El coproducto de una familia de objetos puede no existir. Eso sí, como se desprende de la propia definición:

Teorema 1.2.31 *Si existe el coproducto de una familia de objetos es único salvo isomorfismo.*

Como caso particular de coproducto se tiene la noción de objeto inicial.

Definición 1.2.32 *Sea $\mathcal{C}$ una categoría. Un objeto I se dice* objeto inicial en $\mathcal{C}$ *si, para cualquier otro objeto X de $\mathcal{C}$, el conjunto de morfismos $Morf_{\mathcal{C}}(I, X)$ está formado por un único morfismo. A este morfismo se le denomina* morfismo inicial de X.

Fijarse que el objeto inicial de una categoría corresponde al coproducto de la familia vacía de objetos.

Todo concepto categórico tiene su correspondiente concepto dual, que se obtiene de cambiar el sentido de todas las flechas. La noción dual de la de objeto inicial es la de objeto final, que será un caso particular de producto (dual de coproducto).

Definición 1.2.33 *Sea $\mathcal{C}$ una categoría. Un objeto F se dice* objeto final en $\mathcal{C}$ *si, para cualquier otro objeto X de $\mathcal{C}$, el conjunto de morfismos $Morf_{\mathcal{C}}(X, F)$ está formado por un único morfismo. A este morfismo se le denomina* morfismo final de X.

Resultados conocidos en la Teoría de Categorías, casos particulares del último teorema y de su dual, y que emplearemos, son los recogidos en el siguiente teorema

Teorema 1.2.34

i) Si en una categoría $\mathcal{C}$ existe un objeto inicial, éste es único salvo isomorfismo.

ii) Si en una categoría $\mathcal{C}$ existe un objeto final, éste es único salvo isomorfismo.

Ejemplo 1.2.35 La categoría de conjuntos, la de grupos y la de grupos abelianos poseen objetos inicial y final. En la categoría de conjuntos, el objeto inicial es el conjunto vacío y el final el conjunto unipuntual. En la categoría de grupos y en la de grupos abelianos, ambos, el objeto final y el inicial, son el grupo trivial. En una categoría "slice" $\mathcal{C}/_A$, el morfismo identidad de A en $\mathcal{C}$ es el objeto final.

□

La noción de familia final de objetos generaliza la dada anteriormente.

Definición 1.2.36 *Sea J un conjunto y $\mathcal{C}$ una categoría. Una familia $\{O_j\}_{j\in J}$ de objetos de $\mathcal{C}$ se dice* familia final *si para cualquier objeto A de la categoría, existen un único $j \in J$ y un único morfismo $f\colon A \to O_j$.*

Así, si $\{\mathcal{C}_j\}_{j\in J}$ es una familia de categorías de forma que en cada una de ellas existe objeto final, denotado por O_j, y que constituyen una partición de la categoría $\mathcal{C}$, entonces la familia $\{O_j\}_{j\in J}$ es familia final en $\mathcal{C}$.

Es claro que las equivalencias entre categorías conservan todas las construcciones anteriores (coproductos, objetos y familias finales, objeto inicial). Es decir, la imagen por una equivalencia del coproducto de una familia es el coproducto de la familia de sus imágenes, la imagen del objeto final es final, etc.

1.2.3 Especificación Algebraica

A continuación introducimos algunas nociones básicas sobre especificación algebraica. En [72], [40], [110], [123] pueden encontrarse tal cual las presentamos aquí o con ligeras variantes, dependiendo del formalismo concreto al que se vayan a aplicar.

1.2.3.1 Signaturas

Definición 1.2.37 *Una* signatura Σ *es un par de conjuntos $\langle G, \Omega\rangle$ cuyos elementos se denominan* géneros *y* operaciones*, respectivamente.*

Cada operación de Ω es una $(n+2)$-tupla $\sigma\colon g_1 \dots g_n \to g$, con $g_1, \dots, g_n, g$ géneros y $n \geq 0$; σ es el nombre de la operación*; $g_1, \dots, g_n$; g su* aridad*; $g_1, \dots, g_n$ los* géneros argumento *de la operación y g el* género resultado*. Si $n = 0$, la operación $\sigma : \to g$ se dice* constante de género g*.*

En el caso de que los nombres de todas las operaciones de una signatura sean distintos, abreviaremos la notación y nos referiremos a ellas únicamente por el nombre.

Una signatura es un objeto puramente sintáctico. La idea intuitiva que subyace en ella es que un género denota un conjunto y cada operación denota una función entre los conjuntos denotados por los géneros. Lo natural en una signatura es que todos los géneros aparezcan en

la aridad de alguna operación ya que, en caso contrario, su significado dentro de la signatura es vacío. Si la signatura posee un único género hablaremos de *signatura homogénea*. En caso contrario, hablaremos de *signatura heterogénea*.

Ejemplo 1.2.38

1. Si pensamos en modelar los valores booleanos una posibilidad es utilizar la signatura `Bool`, con un único género que denotamos por *bool* y las siguientes operaciones:

$$\begin{array}{lcccccc} true & : & & & \to & bool \\ false & : & & & \to & bool \\ not & : & bool & & \to & bool \\ and & : & bool & bool & \to & bool \end{array}$$

   La signatura anterior es una signatura homogénea.

2. En el caso de pensar en monoides como estructuras algebraicas, debemos dar el conjunto subyacente, una operación binaria entre los elementos del conjunto y una constante que actua como elemento neutro. Así, pensaríamos en una signatura homogénea `Mnd` con un único género g y las siguientes operaciones:

$$\begin{array}{lcccccc} prd & : & g & g & \to & g \\ unt & : & & & \to & g \end{array}$$

   La signatura anterior es una signatura homogénea.

3. Para trabajar con las acciones de un monoide, utilizaríamos la signatura heterogénea `AccMnd` con dos géneros y las siguientes operaciones:

$$\begin{array}{lcccccc} prd & : & g & g & \to & g \\ unt & : & & & \to & g \\ acc & : & g & c & \to & c \end{array}$$

4. Si nos planteamos modelar los grupos como estructuras algebraicas, debemos pensar en un conjunto base para el grupo y tres operaciones que correspondan con el elemento neutro (constante), el inverso y el producto. La parte sintáctica la formaría la signatura que denotamos por `GRP` con un único género que denotamos por g y las siguientes operaciones:

$$\begin{array}{lcccccc} prd & : & g & g & \to & g \\ inv & : & g & & \to & g \\ unt & : & & & \to & g \end{array}$$

□

Pese a que los ejemplos anteriores han venido motivados por medio de objetos reales, es preciso tener en cuenta que cada signatura solo representa la parte sintáctica.

Como es natural, una vez introducido un nuevo tipo de objetos, las signaturas, definimos la noción de morfismo entre ellas.

Definición 1.2.39 *Sean* $\Sigma = \langle G, \Omega\rangle$ *y* $\Sigma' = \langle G', \Omega'\rangle$ *dos signaturas. Un* morfismo de signaturas $\mu\colon \Sigma \to \Sigma'$ de Σ en Σ' *es un par de funciones totales* $\mu = (\mu_G, \mu_\Omega)$ *con* $\mu_G\colon G \to G'$ *y* $\mu_\Omega\colon \Omega \to \Omega'$ *tal que, para cada operación* σ *de* Ω *con aridad* $g_1, \ldots, g_n; g$*, la operación* $\mu_\Omega(\sigma)$ *de* Ω' *tiene aridad* $\mu_G(g_1), \ldots, \mu_G(g_n); \mu_G(g)$*.*

Signaturas y morfismos de signaturas definen una categoría que denotamos por *Sign*.

Un ejemplo importante de morfismos de signaturas lo constituyen los renombrados. Un *renombrado* es un morfismo de signaturas cuyas funciones son, ambas, biyectivas. Los isomorfismos en la categoría de signaturas coinciden justamente con los renombrados.

Si $\Sigma = \langle G, \Omega\rangle$ y $\Sigma' = \langle G', \Omega'\rangle$ son dos signaturas tales que $G \subseteq G'$ y $\Omega \subseteq \Omega'$ entonces podemos definir el morfismo inclusión $i\colon \Sigma \to \Sigma'$. En este caso se dice que Σ es *subsignatura* de Σ' y se denota por $\Sigma \subseteq \Sigma'$.

1.2.3.2 Álgebras

Una forma de asignar significado a una signatura es asociar un conjunto a cada género y una función a cada operación. Esto nos lleva a la siguiente definición.

Definición 1.2.40 *Sea* $\Sigma = \langle G, \Omega\rangle$ *una signatura. Una* Σ-álgebra parcial A para Σ *es un par* $A = \langle \underline{A}, \Omega_A\rangle$*, siendo* $\underline{A} = (A_g)_{g\in G}$ *un* G*-conjunto y* Ω_A *un* Ω*-conjunto de funciones parciales* $\{\sigma_A\colon A_{g_1} \times \cdots \times A_{g_n} \to A_g,\ Def(\sigma_A)\}_{(\sigma\colon g_1 \ldots g_n \to g)\in\Omega}$*. Para cada* $\sigma\colon g_1 \ldots g_n \to g$ *operación en* Ω*, la función parcial* $(\sigma_A\colon A_{g_1} \times \cdots \times A_{g_n} \to A_g, Def(\sigma_A))$ *se denomina* interpretación en A de la operación $\sigma\colon g_1 \ldots g_n \to g$*. Por su parte, para cada* $g \in G$*, el conjunto* A_g *se denomina* conjunto soporte para el género g*.*

Si todas las funciones de Ω_A *son totales, hablamos entonces de* Σ-álgebra total*.*

Si $\sigma\colon \to g$ es una constante, se tomará como dominio de la interpretación de σ el conjunto unipuntual $\{*\}$, ya que éste es el producto de la familia vacía de conjuntos. Así, la interpretación de una constante de género g en un álgebra A es un elemento de A_g.

Es bastante habitual llamar operación a la interpretación de una operación en un álgebra.

Ejemplo 1.2.41

1. Para la signatura `Bool` podemos definir la `Bool`-álgebra A con conjunto soporte para el género *bool* el conjunto $\{0, 1\}$ y con la siguiente interpretación de las operaciones: $true_A := 1$; $false_A := 0$; $not_A(m) := m + 1\ (mod\ 2)$ y $and_A(m_1, m_2) := m_1 * m_2$. A es `Bool`-álgebra total.

2. Denotamos por B a la `GRP`-álgebra total definida tomando $\mathbb{Z}$ como conjunto soporte para el género g; la suma de enteros de $\mathbb{Z}$ como interpretación de la operación *prd*; el opuesto de cada entero como interpretación de *inv* y el entero 0 como interpretación de la constante *unt*.

   Para cada $p \in \mathbb{N}$, $p > 1$, denotamos por C_p a la `GRP`-álgebra total definida tomando $\mathbb{Z}$ como conjunto soporte para el género g, la suma en $\mathbb{Z}$ módulo p como interpretación de la operación *prd*, el opuesto módulo p como interpretación de *inv* y 0 como constante.

Para cada $p \in \mathbb{N}$, $p > 1$, a partir de cada GRP-álgebra total C_p, podemos definir una GRP-álgebra D_p parcial considerando el conjunto $< p >= \{0, \ldots, p-1\}$ y tomando como dominios: $< p > \times < p >$ para la interpretación de *prd* y $< p >$ para la de *inv*.

□

Una vez definidas las Σ-álgebras, el siguiente paso es relacionarlas por medio de morfismos que respeten la estructura. Vamos en este caso a distinguir explícitamente entre Σ-álgebras totales y parciales.

Definición 1.2.42 *Sea* $\Sigma = \langle G, \Omega \rangle$ *una signatura y sean* $A = \langle \underline{A},\ \Omega_A \rangle$ *y* $B = \langle \underline{B},\ \Omega_B \rangle$ *Σ-álgebras totales. Un* Σ-morfismo entre álgebras totales $f\colon A \to B$ de A en B *es una familia* $f = (f_g\colon A_g \to B_g)_{g \in G}$ *de funciones totales tales que, para cada* $\sigma\colon g_1 \ldots g_n \to g$ *en* Ω *se verifica que* $f_g(\sigma_A(a_1, \ldots, a_n)) = \sigma_B(f_{g_1}(a_1), \ldots, f_{g_n}(a_n))$ *para cada* $(a_1, \ldots, a_n) \in A_{g_1} \times \cdots \times A_{g_n}$.

Intuitivamente, la condición exigida a los morfismos de Σ-álgebras totales se traduce en que conmuten todos los diagramas inducidos por las operaciones de Σ.

Para cada signatura Σ, las Σ-álgebras totales junto con los Σ-morfismos entre ellas definen una categoría que denominamos *Categoría de Σ-álgebras totales* y que denotamos por $Alg(\Sigma)$.

Para Σ-álgebras parciales aparecen en la literatura múltiples nociones de morfismo. Por ejemplo, en [72] (página 36) podemos encontrar tres de las que aparecen propuestas en otras referencias. La que nosotros adoptamos es la que en la referencia anterior se denomina homomorfismo débil, cuya definición recogemos a continuación.

Definición 1.2.43 *Sea* $\Sigma = \langle G, \Omega \rangle$ *una signatura y sean* $A = \langle \underline{A},\ \Omega_A \rangle$ *y* $B = \langle \underline{B},\ \Omega_B \rangle$ *Σ- álgebras parciales. Un* Σ-morfismo $f\colon A \to B$ de A en B *es una familia* $f = (f_g\colon A_g \to B_g)_{g \in G}$ *de funciones totales verificando, para cada* $\sigma\colon g_1 \ldots g_n \to g$ *en* Ω, *las dos siguientes condiciones:*

i) para cada $(a_1, \ldots, a_n) \in Def(\sigma_A)$ *se tiene que* $(f_{g_1}(a_1), \ldots, f_{g_n}(a_n)) \in Def(\sigma_B)$;

ii) para cada $(a_1, \ldots, a_n) \in Def(\sigma_A)$ *se tiene que,*

$$f_g(\sigma_A(a_1, \ldots, a_n)) = \sigma_B(f_{g_1}(a_1), \ldots, f_{g_n}(a_n))$$

para cada $(a_1, \ldots, a_n) \in A_{g_1} \times \cdots \times A_{g_n}$.

Para cada signatura Σ, las Σ-álgebras parciales junto con los Σ-morfismos entre ellas definen una categoría que denominamos *Categoría de Σ-álgebras parciales* y que denotamos por $PAlg(\Sigma)$.

Las dos categorías de álgebras definidas para cada signatura Σ están relacionadas, de forma que la categoría de álgebras totales $Alg(\Sigma)$ es una subcategoría, no plena, de la de álgebras parciales $PAlg(\Sigma)$.

En la categoría de álgebras totales $Alg(\Sigma)$, los isomorfismos son los Σ-morfismos biyectivos. En $PAlg(\Sigma)$, un Σ-morfismo es isomorfismo si es un Σ-morfismo biyectivo y su inverso también es Σ-morfismo.

Cuando no haya lugar a confusión, hablaremos solo de morfismos entre álgebras y, si está claro el marco en el que estamos, total o parcial, tampoco haremos referencia a él.

El objetivo del trabajo en especificación algebraica es modelar sistemas o programas por medio de álgebras, abstrayendo los detalles concretos de los algoritmos y de la codificación. Los métodos de especificación algebraica surgen como herramientas para especificar formalmente la abstracción de los tipos de datos utilizados en programas y disponer así de una potente herramienta matemática que permita analizarlos. Hay diferentes aproximaciones o métodos, cuyos objetivos son muy similares pero entre los que existen importantes diferencias, especialmente a la hora de establecer un modelo semántico para una especificación. Los trabajos [123] y [6] son trabajos dedicados a explicar distintos métodos de especificación algebraica.

La idea en la que se basa la especificación algebraica es modelar estructuras de datos por medio de nombres asociados a los distintos conjuntos de datos que aparecen, de operaciones asociadas a las distintas funciones y de fórmulas asociadas a las propiedades que las estructuras poseen. En general, lo anterior queda recogido en la noción de signatura. Las signaturas constituyen la "parte sintáctica" de las especificaciones y en casi todas las aproximaciones aparecen, si bien también existen distintos tipos de signaturas (véanse distintas variantes en [40], [46], [92] y [55]). Sin embargo, donde realmente se encuentra la diferencia entre las distintas aproximaciones es en la "parte semántica". Tal y como aparece en [109] hay al menos tres niveles distintos para completar una signatura y obtener una especificación, es decir, algo con significado:

- Lo que allí se denomina *semántica de presentación*, donde se considera una especificación como una signatura y un conjunto de axiomas. En este caso, una especificación es algo puramente sintáctico.
- *Semántica de teorías* en la que una especificación está formada por una signatura y un conjunto de axiomas cerrado en una lógica. En este caso el significado de la especificación no es exclusivamente sintáctico.
- *Semántica basada en modelos* en la que una especificación es una signatura junto con una clase de álgebras para esa signatura.

Los distintos métodos de especificación algebraica surgen de las distintas formas de asociar una semántica a una signatura para obtener lo que hemos denominado especificación (y que en la literatura aparece a veces bajo la noción de *Tipo Abstracto de Datos* ([123], [92], [40], [72], [54]).

La semántica que nosotros vamos a utilizar es del tercer tipo. Dentro de las semánticas basadas en modelos, existen distintas nociones de especificación que varían dependiendo de la clase de álgebras que se considera.

La relación entre la Teoría de Categorías y la Especificación Algebraica es estrecha. En [38] y [37] se exploran algunas de esas relaciones.

Entre las aproximaciones semánticas basadas en modelos de las especificaciones algebraicas [92] destacamos: la inicial, en la que el modelo definido por una especificación es el objeto inicial de una categoría [50], [51], [40]; la final, en la que es el objeto final de una adecuada categoría el que proporciona la semántica [121], [91]; la apoximación laxa, en la que la semántica viene dada por toda una categoría de álgebras [92]; y en la observacional (o comportamental) en la que la semántica se da mediante una determinada subcategoría propia de álgebras [92].

Capítulo 2

Especificación algebraica de las estructuras de datos en el sistema EAT

2.1 Las estructuras de datos en EAT

Los procesos de cálculo en Topología Algebraica y en Álgebra Homológica requieren, a diferencia de lo que generalmente ocurre en el Cálculo Simbólico en otras áreas de las Matemáticas (en Geometría Algebraica, por ejemplo), la creación y manipulación, en tiempo de ejecución, de estructuras algebraicas complejas: grupos, conjuntos simpliciales, complejos de cadenas, etc. Este hecho hace que los programas de Cálculo Simbólico en Topología Algebraica deban manejar, implícita o explícitamente, tipos cuyos datos codifiquen a estructuras algebraicas como las citadas. Tenemos entonces datos de dos niveles diferentes: estructuras algebraicas y elementos de esas estructuras. Queremos volver a señalar que lo novedoso no es esto último, sino que esas estructuras tienen que poder ser construidas en tiempo de ejecución, no solo en el momento de la definición.

El programa EAT fue diseñado para el cálculo de la homología de espacios de lazos iterados. En este trabajo no nos vamos a centrar en la explicación de ningún algoritmo concreto. Nuestro interés está en estudiar de un modo formal (matemático) las estructuras de datos usadas en EAT, como paso previo para el análisis global del programa. Veremos que las implementaciones utilizadas en EAT corresponden a un mismo patrón y que poseen buenas propiedades (del tipo de generalidad y universalidad en el sentido de la Teoría de Categorías).

Para comenzar, en esta sección vamos a mostrar cómo son las estructuras de datos que realmente el sistema EAT maneja. Para ello, ilustraremos nuestros comentarios con el ejemplo de la estructura *Grupo* que, pese a ser más simple que los tipos de datos que aparecen en dicho sistema, nos servirá para mostrar las peculiaridades que poseen las estructuras de datos realmente utilizadas.

Supongamos que en uno de los cálculos realizados por EAT para el cómputo de un determinado invariante algebraico de un espacio topológico, se necesita construir un grupo. Un grupo es una estructura algebraica cuyas propiedades vienen dadas axiomáticamente (ecuacionalmente),

ver, por ejemplo, en [71] y que tiene asociada la siguiente signatura que llamamos `GRP`

$$\begin{array}{lcl} prd & : & g \;\; g \rightarrow g \\ inv & : & g \;\;\;\; \rightarrow g \\ unt & : & \;\;\;\;\;\; \rightarrow g \end{array}$$

que recoge las aridades de las tres operaciones que definen un grupo: una operación binaria (el "producto"), una operación unaria (el "inverso") y una constante (el "elemento neutro"). La signatura `GRP` es la base para una especificación algebraica de un grupo, cuyo conjunto subyacente se abstrae mediante el género g.

Para que un sistema de software pueda construir en tiempo de ejecución un grupo, debe incorporar, implícita o explícitamente, un tipo en el que cada uno de sus datos represente a un grupo. Cada dato nos debe permitir acceder a la información asociada a un grupo, es decir, como mínimo recuperar sus operaciones.

Observar que fijando un *universo de datos*, un álgebra queda determinada por sus operaciones. Por tanto, para tener un grupo es suficiente con disponer de sus tres operaciones. Ésta es la idea que se siguió en EAT para la codificación de estructuras algebraicas en general.

Según lo anterior, un tipo registro con tantos campos funcionales como operaciones posea la estructura algebraica a representar (tres en el caso de grupo), nos permite almacenar información suficiente sobre dicha estructura y, por tanto, es un patrón adecuado para su codificación. En cada uno de los campos se almacenará una de las operaciones del álgebra.

La elección de este tipo de estructuras de datos es un primer aspecto en el que se pone de manifiesto la necesidad de utilizar programación funcional en la implementación. Así, no es sorprendente que se usase un lenguaje de naturaleza funcional como Common Lisp para desarrollar EAT.

Una estructura de datos de las comentadas anteriormente y que nos sirve para codificar grupos (en Common Lisp) se define del siguiente modo:

```
(defstruct  grp  prd  inv  unt)
```

donde `grp` es el nombre que se da a la estructura de datos así definida, y `prd`, `inv` y `unt` son los nombres de cada uno de los tres campos (en los que, en nuestro caso, se almacenarán las tres operaciones de grupo). Es claro que cada dato de la estructura anterior permite recuperar un grupo, o por lo menos, la parte esencial del grupo, sus operaciones (siempre que los tres campos almacenen códigos que definan las tres operaciones de un grupo).

La estructura `grp` definida anteriormente, corresponde con la estructura de datos realmente utilizada en el sistema EAT para representar grupos en el computador. Ahora bien, es obvio que pueden definirse otros "tipos Lisp" que también sirvan para el mismo propósito.

A lo largo de este trabajo justificaremos la elección del tipo de estructuras utilizadas en EAT: los registros con campos funcionales. Veremos que este tipo de estructuras son las estructuras de datos más generales entre todas las que pueden definirse para codificar una clase de estructuras algebraicas "del mismo tipo" (con la misma signatura). Es decir, desde cualquier otro modo de implementación siempre se puede construir un paso hacia las implementaciones utilizadas en el sistema EAT. Para estudiar ésta y otras propiedades, emplearemos el marco de la Teoría de Categorías. En este capítulo de la memoria vamos a trabajar a un nivel teórico para: establecer el tipo de signaturas con el que corresponden los tipos usados en EAT, determinar las categorías

de modelos que interesan desde el punto de vista de las necesidades de la implementación, interpretar estas categorías desde distintas perspectivas teóricas y, por último, establecer en estas categorías de modelos la existencia de objetos con propiedades de universalidad que, ya en el siguiente capítulo, se probará que están íntimamente relacionados con las implementaciones usadas en EAT.

Retomando la definición de la estructura de datos `grp`, podemos preguntarnos qué modelos puede representar. Hay una primera respuesta: puede representar a toda una familia de grupos. Ahora bien, no a cualquier familia de grupos, sino a todos los grupos cuyos elementos se encuentran en un cierto universo de datos prefijado. Podríamos decir entonces, que el tipo representa a una familia de grupos sobre un determinado universo o conjunto. Pensando que ese universo de datos forma parte de la máquina (vive en la máquina), podemos concluir que cada dato de la estructura anterior no es un grupo sino, más bien, una implementación de un grupo. Por tanto, siendo más rigurosos, la estructura de datos `grp`, realmente, representa a una familia de *implementaciones de grupos*, ya que lo que manipula no son grupos en sí, sino codificaciones de éstos.

Para que varios grupos puedan ser manipulados en el mismo tipo, todavía hay un ingrediente que está perdido, el tipo de los grupos sobre g está presente pero invisible en la signatura `GRP`. Desde el punto de vista de que `grp` está modelando a una familia de grupos sobre un conjunto (el universo de datos), podemos asociarle la siguiente signatura, que denotamos por `EAT-GRP`:

$$\begin{array}{lcllll} prd\_grp & : & grp & elm & elm & \rightarrow & elm \\ inv\_grp & : & grp & elm & & \rightarrow & elm \\ unt\_grp & : & grp & & & \rightarrow & elm \end{array}$$

Como se ve, esta signatura está muy relacionada con la signatura `GRP`.

En `EAT-GRP`, con el género *grp* se pretende representar a una familia de grupos, mientras que el género *elm* representa el universo de datos, en el cual se encuentran los elementos de los grupos de la familia. La idea es que este universo esté fijado de antemano. Hablando con propiedad, se pretende que el género *grp* represente a las implementaciones de los grupos, mientras que el género *elm*, heredado de la signatura `GRP`, sea utilizado para representar a los elementos (datos) de esas implementaciones de grupo. Observar que las operaciones de la signatura `EAT-GRP` no relacionan distintas implementaciones de grupos entre sí, sino que solo operan dentro de la misma implementación. Notar también cómo se pone de manifiesto la existencia de datos de dos diferentes niveles: estructuras algebraicas (representadas por `grp`) y elementos de esas estructuras (representados por `elm`).

En las siguientes secciones demostraremos que un cierto modelo obtenido de modo natural a partir de la signatura anterior (y que se corresponde con las estructuras de datos utilizadas en EAT) es el más general posible dentro de una adecuada clase de `EAT-GRP`-álgebras. Comentar que en este capítulo nos quedaremos en un nivel teórico: solo nos preocuparemos de aspectos relacionados con la especificación y no de las propias estructuras en la máquina, por lo que trabajaremos a nivel de álgebras.

Otra observación que, aunque obvia, debemos hacer es que las signaturas anteriores `GRP` y `EAT-GRP` se corresponden, no solo con grupos y con familias de grupos, respectivamente, sino que hay otras muchas álgebras para esas signaturas.

Las dos signaturas anteriores, la signatura de grupo `GRP` y la de las familias de grupos `EAT-GRP`, pueden completarse obteniendo especificaciones que realmente representen a los grupos

(añadiendo las ecuaciones necesarias). Sin embargo, en ambos casos, la semántica inicial que es la empleada en el marco clásico de Especificación Algebraica ([40], [72], [6] y [122], entre otros) no proporciona nada de interés para ninguna de las dos signaturas anteriores. Por ejemplo, en el caso de los grupos el modelo inicial es el grupo trivial. Tampoco la semántica final, utilizada en otras aproximaciones (por ejemplo, en [121] y [89]) proporciona nada de interés. Este tipo de signaturas, que provienen del álgebra universal, pese a ser usadas como primeros ejemplos para ilustrar el concepto de signatura (ver, por ejemplo [40], [72]) no son consideradas como base para los Tipos Abstractos de Datos, debido a que el punto de vista habitual en el marco de los Tipos Abstractos de Datos es la semántica inicial que lleva a modelos triviales para este tipo de signaturas.

Dada una signatura, la forma de considerar solo las álgebras que interesen en cada caso y no toda la clase de álgebras, será abordado en posteriores secciones. Nuestro enfoque va a ser entonces el basado en modelos (ver en preliminares, página 27), como en [72] (capítulo 2) en lugar de un enfoque axiomático. Sin embargo, cada construcción y cada resultado que mostremos, puede adaptarse fácilmente al caso axiomático.

La organización de este capítulo es la que comentamos a continuación. En la siguiente sección definimos una operación entre signaturas, la operación $()_{imp}$, que extendemos a las álgebras, para en las siguientes secciones dar distintas interpretaciones, desde diferentes marcos de especificación, de la operación anterior. Así, en la sección 2.3 interpretamos la operación anterior en el marco de la Teoría de Categorías, extrayendo algunas propiedades universales, para en la sección 2.4 trasladarlas a las categorías de álgebras que nos interesan, determinadas subcategorías de las categorías definidas por medio de la operación $()_{imp}$. En la sección 2.5 introducimos otro formalismo de especificación algebraica, las especificaciones ocultas [46] y caracterizamos en este marco las signaturas obtenidas por aplicación de la operación $()_{imp}$. Estudiamos en el marco oculto la existencia de objetos con propiedades universales que trasladamos a nuestro caso, concretamente estudiamos propiedades de coproducto y finalidad. En la sección 2.6 damos otro enfoque a nuestra construcción desde el punto de vista coalgebraico [91], otro formalismo de especificación. En la siguiente sección trasladamos a los Tipos Abstractos de Datos las propiedades estudiadas en las secciones anteriores, pasando a trabajar en el marco parcial en la sección 2.8. El capítulo termina con unas aplicaciones a casos concretos, cercanos a los utilizados en algunos de los cálculos realizados por EAT, y que nos permitirán ir un poco más allá de lo mostrado en el resto del capítulo obteniendo algunas conclusiones que quedan recogidas en el apartado 2.9.4.

En las primeras secciones de este capítulo trabajamos en el marco total, es decir, todas las álgebras consideradas son totales y las funciones también. En posteriores secciones extenderemos lo visto en las primeras al marco parcial.

2.2 Especificación de familias de álgebras

2.2.1 Una operación entre signaturas

En la sección anterior nos han aparecido dos signaturas, `GRP` y `EAT-GRP`, que hemos visto están muy relacionadas. Vamos a comenzar por mostrar cómo la relación entre `GRP` y `EAT-GRP` es solamente un caso particular de una operación general entre signaturas.

Sea SIG el conjunto de signaturas sobre un alfabeto concreto. Definimos $()_{imp}\colon SIG \to SIG$ la aplicación que a cada signatura $\Sigma = \langle G, \Omega \rangle$, le asocia la signatura $\Sigma_{imp} = \langle G_{imp}, \Omega_{imp} \rangle$

definida por:

- $G_{imp} = \{imp_\Sigma\} \cup G$, siendo imp_Σ un género (símbolo) que no está en G. Así, el conjunto G_{imp} de géneros de la nueva signatura está formado por los géneros de Σ y por el nuevo símbolo imp_Σ.
- $\Omega_{imp} = \{imp\_\sigma \colon imp_\Sigma\ g_1 \dots g_n \to g\}_{(\sigma \colon g_1 \dots g_n \to g) \in \Omega}$

  Por cada operación $\sigma \colon g_1 \dots g_n \to g$ en Ω, se incluye en Ω_{imp} una operación $imp\_\sigma \colon imp_\Sigma\ g_1 \dots g_n \to g$.

Obsérvese que Ω_{imp} tiene el mismo número de operaciones que Ω. Además, en cada una de las operaciones de Ω_{imp} aparece el nuevo género imp_Σ una única vez y siempre como primer argumento. La observación anterior, nos permite afirmar que, considerando imp_Σ como género distinguido de la signatura Σ_{imp}, todas las operaciones de la signatura son observadoras, según la terminología utilizada habitualmente en el ámbito de la especificación algebraica (ver [40] y [72], entre otros).

Ejemplo 2.2.1 Si a la signatura GRP

$$\begin{array}{lcl} prd & : & g\ \ g \to g \\ inv & : & g \to g \\ unt & : & \to g \end{array}$$

se le aplica la operación $()_{imp}$, la signatura resultante es la signatura $\mathtt{GRP}_{imp}$

$$\begin{array}{lcl} imp\_prd & : & imp_{\mathtt{GRP}}\ \ g\ \ g \to g \\ imp\_inv & : & imp_{\mathtt{GRP}}\ \ g \to g \\ imp\_unt & : & imp_{\mathtt{GRP}} \to g \end{array}$$

que, por renombrado de géneros y operaciones (véanse, preliminares página 25, o [72] página 71), es isomorfa a la signatura EAT-GRP, de la sección anterior.

□

2.2.2 Interpretación informal de las Σ_{imp}-álgebras como familias de Σ-álgebras

La operación $()_{imp}$ está definida únicamente entre signaturas, es decir, en un nivel sintáctico. La única información de la que se dispone es el número de géneros y las aridades de las operaciones.

Nuestro siguiente objetivo es estudiar la categoría de Σ_{imp}-álgebras y ver qué tipo de relación existe entre ella y la categoría de Σ-álgebras. Recordemos que dada una signatura Σ, definir una Σ-álgebra consiste en asociar, a cada género de la signatura, un conjunto que se denomina *soporte* para ese género, y definir, para cada operación de la signatura, una función entre los correspondientes soportes.

Veamos algunas álgebras para la signatura $\mathtt{GRP}_{imp}$ del ejemplo anterior.

Ejemplo 2.2.2

1. Llamamos A a la $\mathtt{GRP}_{imp}$-álgebra que tiene a $\mathbb{Z}$ como soporte para los dos géneros y como operaciones:

$$
\begin{array}{lcccccccc}
imp\_prd_A & : & \mathbb{Z} & \times & \mathbb{Z} & \times & \mathbb{Z} & \to & \mathbb{Z} \\
 & & (z & , & z1 & , & z2) & \mapsto & z1 \\
imp\_inv_A & : & \mathbb{Z} & \times & \mathbb{Z} & \to & \mathbb{Z} & & \\
 & & (z & , & z1) & \mapsto & -z1 & & \\
imp\_unt_A & : & \mathbb{Z} & \to & \mathbb{Z} & & & & \\
 & & z & \mapsto & 0 & & & &
\end{array}
$$

Así definida A es $\texttt{GRP}_{imp}$-álgebra puesto que las aridades de sus operaciones se corresponden con las de la signatura.

El concepto de álgebra para una signatura es muy poco exigente. En particular, el álgebra A dada en este ejemplo no corresponde a ninguna estructura que nos interese. Los siguientes ejemplos recogen Σ_{imp}-álgebras más significativas desde el punto de vista del Álgebra [71].

2. Definimos la $\texttt{GRP}_{imp}$-álgebra B en la que $\mathbb{Z}$ es el soporte para el género g y $\mathbb{N}$ el correspondiente al género distinguido $imp_{\texttt{GRP}}$. Las operaciones las definimos del siguiente modo:

   $imp\_prd_B(n, z1, z2) = z1 + z2$, para todo $n \in \mathbb{N}$ y todo $z1, z2 \in \mathbb{Z}$.
   $imp\_inv_B(n, z) = -z$, para todo $n \in \mathbb{N}$ y $z \in \mathbb{Z}$.
   $imp\_unt_B(n) = 0$, para todo $n \in \mathbb{N}$.

   En la definición de esta $\texttt{GRP}_{imp}$-álgebra subyace la idea de representar con ella una familia de GRP-álgebras. En este sentido, observar que para cada $n \in \mathbb{N}$, la $\texttt{GRP}_{imp}$-álgebra B, define una GRP-álgebra, $\langle \mathbb{Z}, \{+_{\mathbb{Z}}, -_{\mathbb{Z}}, 0_{\mathbb{Z}}\}\rangle$, que es un grupo como estructura algebraica. Así, se puede interpretar la $\texttt{GRP}_{imp}$-álgebra B como una colección de copias, indexada por el conjunto de los números naturales, del grupo $\mathbb{Z}$. Por tanto, en este ejemplo sí hay una idea de estructura debajo de la definición de la $\texttt{GRP}_{imp}$-álgebra. Esta idea de interpretar una $\texttt{GRP}_{imp}$-álgebra como una familia de GRP-álgebras es la que intentaremos explotar más adelante.

3. Consideramos la $\texttt{GRP}_{imp}$-álgebra C con el mismo soporte que en el ejemplo anterior para el género g y $\mathbb{N} \setminus \{0, 1\}$ para el género $imp_{\texttt{GRP}}$, y cuyas operaciones vienen dadas por:

   $imp\_prd_C(n, z1, z2) = (z1 + z2) \bmod n$, para todo $n \in \mathbb{N} \setminus \{0, 1\}$ y todo $z1, z2 \in \mathbb{Z}$.
   $imp\_inv_C(n, z) = -z \bmod n$, para todo $n \in \mathbb{N} \setminus \{0, 1\}$ y $z \in \mathbb{Z}$.
   $imp\_unt_C(n) = 0$, para todo $n \in \mathbb{N} \setminus \{0, 1\}$.

   En este caso, cada $n \in \mathbb{N}$ con $n > 1$, determina una GRP-álgebra que denotamos por C_n y que viene dada por $C_n = \langle \mathbb{Z}, \{+ \bmod n, - \bmod n, 0_{\mathbb{Z}}\}\rangle$. Fijado un $n \in \mathbb{N}$, $n > 1$, C_n *tiene el mismo comportamiento* que el grupo $\langle \mathbb{Z}/n\mathbb{Z}, \{+_{\mathbb{Z}/n\mathbb{Z}}, -_{\mathbb{Z}/n\mathbb{Z}}, 0_{\mathbb{Z}/n\mathbb{Z}}\}\rangle$. En la expresión *mismo comportamiento* queremos recoger que pese a que los soportes son distintos, las operaciones ante elementos "similares" en $\mathbb{Z}$, soporte de C_n, y $\mathbb{Z}/n\mathbb{Z}$ devuelven elementos "similares". Más aún, en C_n es posible elegir representantes canónicos, por ejemplo el conjunto $< n >= \{0, 1, \ldots, n-1\}$. En particular, dicha noción implica que la aplicación cociente $p\colon \mathbb{Z} \to \mathbb{Z}/n\mathbb{Z}$ define un morfismo entre ambas GRP-álgebras, es decir, el paso al cociente es compatible con las operaciones de las álgebras C_n. Sin embargo, la noción de paso al cociente es más débil que la de tener el mismo comportamiento: para cada $n \in \mathbb{N}$, el morfismo $p\colon \mathbb{Z} \to \mathbb{Z}/n\mathbb{Z}$ entre B_n (siendo B la Σ_{imp}-álgebra del ejemplo

anterior) y $\langle \mathbb{Z}/n\mathbb{Z}, \{+_{\mathbb{Z}/n\mathbb{Z}}, -_{\mathbb{Z}/n\mathbb{Z}}, 0_{\mathbb{Z}/n\mathbb{Z}}\}\rangle$ es un paso al cociente y sin embargo, ambas GRP-álgebras no tienen el mismo comportamiento. Por tanto, podemos ver C como una *representación* de la familia de GRP-álgebras $\{\mathbb{Z}/n\mathbb{Z}\}_{n\in\mathbb{N},n>1}$. Las nociones de comportamiento y representación, formalizadas de modo conveniente, serán importantes en el resto del trabajo.

4. Consideramos la misma signatura $\mathtt{GRP}_{imp}$ y para cada $n \in \mathbb{N}$ fijo consideramos el conjunto $< n >= \{0, 1, \ldots, n-1\}$. Definimos la $\mathtt{GRP}_{imp}$-álgebra D tomando $< n >$ como soporte para el género g y, para el género $imp_{\mathtt{GRP}}$, el conjunto $D_{imp_{\mathtt{GRP}}} = \{(n_1, \ldots, n_k) \mid k \geq 1, n_i \in \mathbb{N}, n_i > 1, \forall i = 1, \ldots, k \ \wedge \ n = n_1 * \ldots * n_k\}$. La idea es que cada tupla $(n_1, \ldots, n_k)$ de $D_{imp_{\mathtt{GRP}}}$ representa al grupo abeliano finito $\mathbb{Z}/n_1\mathbb{Z} \times \cdots \times \mathbb{Z}/n_k\mathbb{Z}$. Por tanto, el conjunto $D_{imp_{\mathtt{GRP}}}$ está dando soporte a una familia de grupos abelianos finitos.

   Para completar la definición de la $\mathtt{GRP}_{imp}$-álgebra D hay que introducir las operaciones. Teniendo en cuenta la última observación, las operaciones de D se definirán a partir de las de cada grupo $\mathbb{Z}/n_1\mathbb{Z} \times \cdots \times \mathbb{Z}/n_k\mathbb{Z}$ y, por tanto, será suficiente con dar una biyección entre el conjunto $< n >$ y los elementos del grupo $\mathbb{Z}/n_1\mathbb{Z} \times \cdots \times \mathbb{Z}/n_k\mathbb{Z}$. Cada elemento del grupo $\mathbb{Z}/n_1\mathbb{Z} \times \cdots \times \mathbb{Z}/n_k\mathbb{Z}$ puede representarse por una tupla $(a_1, \ldots, a_k)$ con $a_i \in \{0, \ldots, n_i - 1\}$, para todo $i = 1, \ldots, k$. Consideramos la biyección $enum_{(n_1,\ldots,n_k)} : \mathbb{Z}/n_1\mathbb{Z} \times \cdots \times \mathbb{Z}/n_k\mathbb{Z} \to \{0, 1, \ldots, n-1\}$ que asocia a cada tupla $(a_1, \ldots, a_k) \in \mathbb{Z}/n_1\mathbb{Z} \times \cdots \times \mathbb{Z}/n_k\mathbb{Z}$ el valor

$$enum_{(n_1,\ldots,n_k)}\big((a_1, \ldots, a_k)\big) = \sum_{i=1}^{k}\Big(\prod_{j=i+1}^{k} n_j\Big)a_i$$

   Esta biyección nos permite enumerar los elementos de cada grupo $\mathbb{Z}/n_1\mathbb{Z} \times \cdots \times \mathbb{Z}/n_k\mathbb{Z}$. A partir de ella, las operaciones de D se definen del modo obvio, lo que completa la definición de la $\mathtt{GRP}_{imp}$-álgebra D.

   Por ejemplo, si $n = 12$, la tupla $(2, 2, 3)$ representa al grupo $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/3\mathbb{Z}$. La biyección $enum_{(2,2,3)}$ lleva a cada tupla (i, j, k) con $i, j \in \{0, 1\}$ y $k \in \{0, 1, 2\}$ al natural $6 * i + 3 * j + k \in < n >$. La tupla $(2, 6)$ representa al grupo $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/6\mathbb{Z}$ y $(4, 3)$ a $\mathbb{Z}/4\mathbb{Z} \times \mathbb{Z}/3\mathbb{Z}$. Observar que entre los tres grupos anteriores, los dos primeros son isomorfos.

   Cualquier grupo abeliano finito es isomorfo a uno del tipo $\mathbb{Z}/n_1\mathbb{Z} \times \cdots \times \mathbb{Z}/n_k\mathbb{Z}$, con $n_i \in \mathbb{N}$, para todo $i = 1, \ldots, k$. Por tanto, fijado $n \in \mathbb{N}$, la familia definida en este ejemplo cubre todos los grupos abelianos finitos de cardinal n. Barriendo todos los $n \in \mathbb{N}$, se cubren todos los grupos abelianos finitos. Luego este ejemplo nos aporta un patrón común de representación para los grupos abelianos finitos.

   Además, desde cualquier representación de grupos abelianos definidos sobre $< n >$, existe al menos un $\mathtt{GRP}_{imp}$-morfismo a la $\mathtt{GRP}_{imp}$-álgebra D. Un tal $\mathtt{GRP}_{imp}$-morfismo puede definirse de manera sencilla siempre que los elementos de $< n >$ queden fijos.

5. La $\mathtt{GRP}_{imp}$-álgebra D anterior nos permite representar familias de grupos abelianos finitos. Vamos ahora a dar una $\mathtt{GRP}_{imp}$-álgebra, que llamaremos E y que recoge el caso de los grupos finitos cualesquiera.

   Fijamos un $n \in \mathbb{N}$, tomamos el conjunto $< n >$ como soporte para el género g. Consideramos como soporte para el género $imp_{\mathtt{GRP}}$ el conjunto $E_{imp_{\mathtt{GRP}}} = \{$ matrices $n \times n \mid$ la matriz es la tabla del producto de un grupo sobre $< n >\}$. De cada tabla pueden extraerse los inversos de cada elemento así como los elementos neutros, por lo que la $\mathtt{GRP}_{imp}$-álgebra E representa de forma natural a una familia de grupos.

Cualquier grupo finito es isomorfo a uno de los grupos indexados por el conjunto $E_{imp_{\texttt{GRP}}}$ anterior, por lo que, barriendo todos los $n \in \mathbb{N}$, se cubren todos los grupos finitos. En particular, las matrices que además sean simétricas cubrirán las representaciones de grupos abelianos y las no simétricas las de grupos no abelianos.

Por ejemplo, fijamos $n = 6$ y consideramos el grupo de permutaciones de tres elementos que denotamos por $\mathcal{S}_3$. Sin perder generalidad, podemos suponer que los tres elementos permutados son $1, 2$ y 3. Si asignamos a cada permutación el número de orden que le corresponde entre 0 y 5 (orden definido por la biyección $enum_{\mathcal{S}_3} : \mathcal{S}_3 \rightarrow < n >$ dada por $enum_{\mathcal{S}_3}\big((1\ 2\ 3)\big) = 0$, $enum_{\mathcal{S}_3}\big((1\ 3\ 2)\big) = 1, enum_{\mathcal{S}_3}\big((2\ 1\ 3)\big) = 2, enum_{\mathcal{S}_3}\big((2\ 3\ 1)\big) = 3, enum_{\mathcal{S}_3}\big((3\ 1\ 2)\big) = 4$ y $enum_{\mathcal{S}_3}\big((3\ 2\ 1)\big) = 5$), la siguiente matriz representa la tabla de multiplicar del grupo:

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 1 & 0 & 3 & 2 & 5 & 4 \\ 2 & 4 & 0 & 5 & 1 & 3 \\ 3 & 5 & 1 & 4 & 0 & 2 \\ 4 & 2 & 5 & 0 & 3 & 1 \\ 5 & 3 & 4 & 1 & 2 & 0 \end{pmatrix}$$

Es evidente que de ella se puede extraer el inverso de cada elemento y que la permutación $(1\ 2\ 3)$ es el elemento neutro del grupo.

Desde cualquier representación de grupos definidos sobre $< n >$, existe un único $\texttt{GRP}_{imp}$-morfismo a la $\texttt{GRP}_{imp}$-álgebra E, manteniendo fijo el soporte $< n >$. Por ello, podemos afirmar que la $\texttt{GRP}_{imp}$-álgebra E es la más "general" entre las representaciones de grupos sobre $< n >$. Esto se formalizará en una propiedad de finalidad del álgebra E en una categoría adecuada.

En particular, la $\texttt{GRP}_{imp}$-álgebra D da una representación para grupos abelianos finitos y, para cada $n \in \mathbb{N}$ fijo, el $\texttt{GRP}_{imp}$-morfismo de D a E que mantiene fijo $< n >$ viene dado por una aplicación f que a cada tupla $(n_1, \ldots, n_k) \in D_{imp_{\texttt{GRP}}}$ le asocia una matriz $n \times n$. Por ejemplo, si consideramos $n = 6$, se tiene $D_{imp_{\texttt{GRP}}} = \{(2,3), (3,2), (6)\}$. Para asociar una matriz a cada uno de los elementos de $D_{imp_{\texttt{GRP}}}$ utilizamos la biyección $enum_{(n_1,\ldots,n_k)}$, y así, obtenemos que

$$f_{imp_{\texttt{GRP}}}\big((2,3)\big) = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 0 & 4 & 5 & 3 \\ 2 & 0 & 1 & 5 & 3 & 4 \\ 3 & 4 & 5 & 0 & 1 & 2 \\ 4 & 5 & 3 & 1 & 2 & 0 \\ 5 & 3 & 4 & 2 & 0 & 1 \end{pmatrix} \qquad f_{imp_{\texttt{GRP}}}\big((3,2)\big) = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 1 & 0 & 3 & 2 & 5 & 4 \\ 2 & 3 & 4 & 5 & 0 & 1 \\ 3 & 2 & 5 & 4 & 1 & 0 \\ 4 & 5 & 0 & 1 & 2 & 3 \\ 5 & 4 & 1 & 0 & 3 & 2 \end{pmatrix}$$

□

En los últimos ejemplos, concretamente en los cuatro últimos, que son los que tienen algún "sentido algebraico", se ha puesto de manifiesto cómo una $\texttt{GRP}_{imp}$-álgebra define una familia de GRP-álgebras. Generalizamos el comentario anterior en la siguiente proposición, cuya demostración es evidente.

Previamente introducimos la siguiente notación abreviada que utilizaremos a partir de ahora siempre que no haya lugar a confusión. Dada una signatura Σ, una operación σ de Ω con aridad $g_1 \ldots g_n; v$ será denotada por $(\sigma : \omega \rightarrow v)$, refiriéndonos por ω a la secuencia de géneros $g_1 \ldots g_n$;

análogamente, dada una Σ-álgebra A, A_ω denotará al conjunto $A_{g_1} \times \cdots \times A_{g_n}$. En el caso de que ω sea la secuencia vacía, lo representaremos por $\omega = []$, y, por convenio, consideraremos A_ω como un conjunto unipuntual que denotaremos por $\{*\}$. Dada una Σ_{imp}-álgebra A y una operación $\sigma\colon g_1 \dots g_n \to v$ de Ω, para cada $a \in A_{imp_\Sigma}$, denotaremos por $imp\_\sigma_A(a,-)\colon A_\omega \to A_v$ a la imagen de a por la adjunta exponencial de $imp\_\sigma_A\colon A_{imp_\Sigma} \times A_\omega \to A_v$, es decir, $imp\_\sigma_A(a,-)(d_\omega) = imp\_\sigma_A(a, d_\omega)$.

Proposición 2.2.3 *Si* $A = \langle A_{imp_\Sigma}, (A_g)_{g\in G}, \{imp\_\sigma_A\colon A_{imp_\Sigma} \times A_\omega \to A_v\}_{(\sigma\colon \omega\to v)\in\Sigma}\rangle$ *es una* Σ_{imp}*-álgebra, cada* $a \in A_{imp_\Sigma}$ *define una* Σ*-álgebra* B_a *de la siguiente manera:* $B_a = \langle (A_g)_{g\in G}, \{imp\_\sigma_A(a,-)\colon A_\omega \to A_v\}_{(\sigma\colon \omega\to v)\in\Sigma}\rangle$.

Resaltemos que la familia $\{B_a\}_{a\in A_{imp_\Sigma}}$ de Σ-álgebras que se obtiene en la proposición anterior tiene la siguiente propiedad: los soportes de todas las Σ-álgebras de la familia obtenida a partir de una Σ_{imp}-álgebra, vienen determinados por los de la Σ_{imp}-álgebra de partida; más concretamente, para cada género g de Σ, el soporte de todas las Σ-álgebras de la familia es constante e igual a A_g.

Volviendo a nuestros ejemplos, puede comprobarse que todas las familias de álgebras referidas en ellos han sido construidas del modo recogido en la última proposición. Los soportes para el género imp_Σ juegan el papel de familias de índices y deben interpretarse como modos de representar estructuradamente, bajo un mismo patrón, ejemplares de una misma estructura algebraica. El disponer de un modo común de representación será útil al descender al nivel de las implementaciones. Esta cuestión la abordaremos en el siguiente capítulo.

2.3 Interpretación en términos de Teoría de Categorías

En esta sección veremos cómo la interpretación de las Σ_{imp}-álgebras como familias indexadas se corresponde con una construcción de tipo categorial. Vamos a presentar dicha construcción y a analizar la existencia de ciertos objetos con propiedades universales que, posteriormente, intentaremos trasladar a la categoría objeto de nuestro estudio, la categoría $Alg(\Sigma_{imp})$. Esta categoría será la base para estudiar las estructuras de datos de EAT.

En lo que sigue, si $\mathcal{C}$ es categoría, denotaremos por $Obj(\mathcal{C})$ a la clase de objetos de $\mathcal{C}$ y por $Morf(\mathcal{C})$ a la clase de morfismos de $\mathcal{C}$.

2.3.1 Una operación entre categorías

Dada una categoría $\mathcal{C}$, definimos una nueva categoría, denotada por $\mathcal{C}_{Set}$, que intuitivamente sirve para representar a la categoría de las familias indexadas de objetos de la categoría $\mathcal{C}$.

- Los objetos de $\mathcal{C}_{Set}$ son pares (A, α_A), donde A es un conjunto y $\alpha_A\colon A \to Obj(\mathcal{C})$ es una aplicación.

  Describiremos un objeto por $\alpha_A\colon A \to Obj(\mathcal{C})$. Incluso, si no hay lugar a confusión, nos referiremos al objeto por α_A.

- Dados dos objetos $\alpha_A\colon A \to Obj(\mathcal{C})$ y $\alpha_B\colon B \to Obj(\mathcal{C})$ de $\mathcal{C}_{Set}$, un morfismo de α_A en α_B es un par (h, H) donde $h\colon A \to B$ es una aplicación y, para cada $a \in A$, $H(a)$ es un morfismo en $\mathcal{C}$ de $\alpha_A(a)$ en $\alpha_B(h(a))$.

Así definida, es claro que $\mathcal{C}_{Set}$ es una categoría.

La categoría $\mathcal{C}_{Set}$ corresponde a construcciones conocidas en Teoría de Categorías y suficientemente tratadas en la literatura. Por ejemplo, puede verse como categoría que resulta de "aplanar" una determinada categoría indexada [118]. La categoría $\mathcal{C}_{Set}$ corresponde también con un caso particular de categoría fibrada, según la definición dada en [17]. (Definiciones de categorías indexadas y categorías fibradas, así como estudios de estas categorías pueden encontrarse, por ejemplo, en [17], [43] o [79].) Es también importante señalar que si $\mathcal{C}$ es una categoría discreta (es decir, equivale a la clase de sus objetos), $\mathcal{C}_{Set}$ corresponde justamente a la clase de las familias indexadas de "elementos" de $\mathcal{C}$.

Cada objeto $\alpha_A \colon A \to Obj(\mathcal{C})$ de la categoría $\mathcal{C}_{Set}$ es una familia indexada, por el conjunto A, de objetos de la categoría $\mathcal{C}$, la familia $\{\alpha_A(a)\}_{a \in A}$. A partir de ahora, llamaremos a A *conjunto de índices* del objeto α_A.

En la siguiente proposición se da una caracterización de los isomorfismos en la categoría $\mathcal{C}_{Set}$.

Proposición 2.3.1 *Un morfismo (h, H) en $\mathcal{C}_{Set}$ de un objeto $\alpha_A \colon A \to Obj(\mathcal{C})$ en un objeto $\alpha_B \colon B \to Obj(\mathcal{C})$ es isomorfismo si la aplicación $h \colon A \to B$ es biyectiva y para todo $a \in A$, $H(a)$ es isomorfismo en $\mathcal{C}$.*

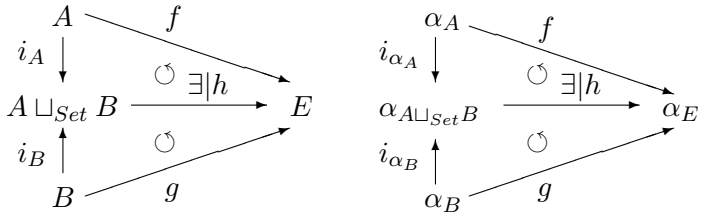
La notación que estamos empleando para referirnos a los objetos de la categoría $\mathcal{C}_{Set}$ puede resultar un poco confusa. Ya que cada conjunto A no determina una aplicación de A en $Obj(\mathcal{C})$, es decir, no determina un objeto de $\mathcal{C}_{Set}$, la notación α_A no es del todo precisa. Sin embargo, la empleamos porque, como veremos más adelante, es la utilizada habitualmente en la aproximación coalgebraica ([91], [106]), y el uso de dicha notación pondrá más en evidencia la relación de las categorías de la forma $\mathcal{C}_{Set}$ con categorías de coálgebras.

2.3.2 Existencia de coproductos en $\mathcal{C}_{Set}$

A continuación, vamos a describir el coproducto de parejas de objetos de $\mathcal{C}_{Set}$.

Dados dos objetos, $\alpha_A \colon A \to Obj(\mathcal{C})$ y $\alpha_B \colon B \to Obj(\mathcal{C})$ de $\mathcal{C}_{Set}$, consideramos el objeto de $\mathcal{C}_{Set}$, denotado por $[\alpha_A, \alpha_B] \colon A \sqcup B \to Obj(\mathcal{C})$, que viene definido por $[\alpha_A, \alpha_B](a) := \alpha_A(a)$, para todo $a \in A$ y $[\alpha_A, \alpha_B](b) := \alpha_B(b)$, para todo $b \in B$.

Como puede verse, la operación anterior a nivel de los conjuntos de índices se corresponde con la operación de reunión disjunta, es decir, con el coproducto en Set. Es sencillo verificar que este objeto de $\mathcal{C}_{Set}$ junto con las inclusiones naturales de α_A y α_B en $[\alpha_A, \alpha_B]$ verifica la propiedad universal de coproducto. Por tanto, la operación $[-,-]$ es el coproducto en $\mathcal{C}_{Set}$ (de ahí el uso de la notación empleada habitualmente en Teoría de Categorías). Así, la categoría $\mathcal{C}_{Set}$ posee coproductos finitos.



Igual que la reunión disjunta se extiende a cualquier familia de conjuntos, el coproducto en $\mathcal{C}_{Set}$ puede extenderse también a familias arbitrarias de objetos. Así, si J es un conjunto cualquiera, a partir de una familia $\{\alpha_{A_j}\colon A_j \to Obj(\mathcal{C})\}_{j\in J}$ consideramos el siguiente objeto de $\mathcal{C}_{Set}$

$$[\alpha_{A_j}]_{j\in J}\colon \bigsqcup_{j\in J} A_j \to Obj(\mathcal{C})$$

dado por $[\alpha_{A_j}]_{j\in J}(a) := \alpha_{A_k}(a)$, siendo k el único índice de J tal que $a \in A_k$.

El objeto $[\alpha_{A_j}]_{j\in J}$ junto con la correspondiente familia de morfismos inclusión define el coproducto general en $\mathcal{C}_{Set}$. Es decir, para cada $j \in J$, consideramos el morfismo $i_{\alpha_{A_j}}\colon \alpha_{A_j} \to [\alpha_{A_j}]_{j\in J}$ de $\mathcal{C}_{Set}$ definido por $i_{\alpha_{A_j}} = (i_{A_j}, H_{i_{A_j}})$ siendo

- $i_{A_j}\colon A_j \to \bigsqcup_{j\in J} A_j$ la inclusión canónica, y
- $H_{i_{A_j}}\colon A_j \to Morf(\mathcal{C})$ la aplicación que a cada elemento a de A_j le asocia el morfismo identidad de $\alpha_{A_j}(a)$.

Se tiene que $\langle [\alpha_{A_j}]_{j\in J}, \{i_{\alpha_{A_j}}\}_{j\in J}\rangle$ es el coproducto de la familia $\{\alpha_{A_j}\colon A_j \to Obj(\mathcal{C})\}_{j\in J}$. Así, se tiene el siguiente resultado:

Teorema 2.3.2 *Sea $\mathcal{C}$ una categoría. La categoría $\mathcal{C}_{Set}$, definida a partir de $\mathcal{C}$ aplicando la operación $()_{Set}$, posee coproductos arbitrarios.*

2.3.3 Interpretación de $\mathcal{C}$ como subcategoría de $\mathcal{C}_{Set}$

Va a existir una forma canónica de incluir a $\mathcal{C}$ como subcategoría de $\mathcal{C}_{Set}$. Cada objeto de $\mathcal{C}_{Set}$ cuyo conjunto de índices tiene un único elemento puede ser interpretado como un objeto de $\mathcal{C}$, lo que nos da una idea de cómo definir una inclusión de $\mathcal{C}$ en $\mathcal{C}_{Set}$.

En Teoría de Categorías, la noción fuerte de inclusión de una categoría $\mathcal{C}$ en otra $\mathcal{D}$ corresponde con la existencia de una *inmersión* de $\mathcal{C}$ en $\mathcal{D}$, es decir, un funtor de $\mathcal{C}$ en $\mathcal{D}$ fiel, pleno e inyectivo sobre objetos (ver en preliminares, página 20). Relajando un poco las condiciones anteriores, si un funtor de $\mathcal{C}$ en $\mathcal{D}$ es fiel e inyectivo sobre objetos diremos que es una *inclusión* de $\mathcal{C}$ en $\mathcal{D}$. La imagen de una categoría $\mathcal{C}$ por cualquier inmersión o inclusión es una subcategoría de $\mathcal{D}$, plena en el caso de inmersión, isomorfa a la categoría $\mathcal{C}$.

Existe una inclusión canónica de $\mathcal{C}$ en $\mathcal{C}_{Set}$ que nos permite ver $\mathcal{C}$ como subcategoría de $\mathcal{C}_{Set}$. Una forma de expresar esta inclusión es definir $F_{\mathcal{C}}\colon \mathcal{C} \to \mathcal{C}_{Set}$ del siguiente modo:

- Para cada O objeto de $\mathcal{C}$, $F_{\mathcal{C}}(O)$ es el objeto de $\mathcal{C}_{Set}$, $F_{\mathcal{C}}(O)\colon \{O\} \to Obj(\mathcal{C})$, dado por $F_{\mathcal{C}}(O)(O) := O$.
- Si O_1 y O_2 son objetos de $\mathcal{C}$ y $f\colon O_1 \to O_2$ es un morfismo en $\mathcal{C}$, definimos $F_{\mathcal{C}}(f)\colon F_{\mathcal{C}}(O_1) \to F_{\mathcal{C}}(O_2)$ como el par $(\{O_1\} \to \{O_2\}, f)$.

2.3.4 Objeto de $\mathcal{C}_{Set}$ que "representa" a toda la categoría $\mathcal{C}$

En el caso de que $\mathcal{C}$ sea una categoría pequeña, es decir, que $Obj(\mathcal{C})$ sea un conjunto, existe un objeto especial en $\mathcal{C}_{Set}$ que "recoge" a toda la categoría $\mathcal{C}$.

En el resto del capítulo, supondremos que $\mathcal{C}$ es una categoría pequeña.

Consideramos el objeto $\mathbb{1}\colon Obj(\mathcal{C}) \to Obj(\mathcal{C})$ de $\mathcal{C}_{Set}$ definido por la aplicación identidad. La exigencia de que $\mathcal{C}$ sea pequeña es necesaria para que $\mathbb{1}$ sea un objeto de $\mathcal{C}_{Set}$. (En realidad, podríamos trabajar con clases pero para nuestro trabajo nos basta con tomar conjuntos, evitando de paso los problemas de fundamentos que pudieran aparecer.)

La familia definida por el objeto $\mathbb{1}$, $\{\mathbb{1}(O)\}_{O\in Obj(\mathcal{C})}$, está formada por todos los objetos de $\mathcal{C}$. Por tanto, $\mathbb{1}$ "codifica" a todos los objetos de la categoría $\mathcal{C}$. Evidentemente, este objeto $\mathbb{1}$ está relacionado con las inclusiones de $\mathcal{C}$ en $\mathcal{C}_{Set}$. El siguiente resultado describe formalmente el hecho de que $\mathbb{1}$ recoge a toda $\mathcal{C}$ como parte de $\mathcal{C}_{Set}$.

Teorema 2.3.3 *Si $\mathcal{C}$ es una categoría pequeña, el objeto $\mathbb{1}$ es el coproducto en $\mathcal{C}_{Set}$ de los objetos de la imagen de la inclusión canónica de $\mathcal{C}$ en $\mathcal{C}_{Set}$.*

El objeto $\mathbb{1}$, a pesar de que su definición es sencilla y aparentemente alejada de cualquier contexto práctico de programación, nos va a servir para establecer una conexión con diferentes teorías de especificación orientada a objeto que aparecen en la literatura (ver [23], [46], [60], [91], [27], [44], [28], [62], entre otros). Como se verá, el objeto $\mathbb{1}$ está muy cercano a objetos finales y a familias de objetos finales que interesan en diferentes ámbitos de la especificación algebraica orientada a objeto.

2.3.5 Objetos finales

Dado cualquier objeto $\alpha_A\colon A \to Obj(\mathcal{C})$ de $\mathcal{C}_{Set}$, la propia aplicación α_A junto con la familia de identidades de los objetos $\{\alpha_A(a)\}_{a\in A}$ define un morfismo en $\mathcal{C}_{Set}$ de α_A en $\mathbb{1}$. Este morfismo lo denotaremos por el par $(\alpha_A, id_{\alpha_A})$. Esta observación se formaliza en el siguiente resultado.

Proposición 2.3.4 *Existe, al menos, un morfismo de cualquier objeto de $\mathcal{C}_{Set}$ en $\mathbb{1}$.*

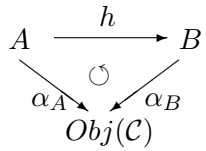
Ahora bien, el objeto $\mathbb{1}$ no es final en $\mathcal{C}_{Set}$ ya que en general, hay más de un morfismo entre un objeto cualquiera α_A de $\mathcal{C}_{Set}$ y $\mathbb{1}$. Los grados de libertad a la hora de definir estos posibles morfismos vienen relacionados con dos aspectos. Por un lado, con la definición de la aplicación entre los conjuntos de índices, puesto que no necesariamente ha de utilizarse, como en el morfismo definido anteriormente, la propia aplicación α_A. Por otro lado, con los conjuntos de endomorfismos $Morf_{\mathcal{C}}(\alpha_A(a), \alpha_A(a))$. Si (h, H) es morfismo en $\mathcal{C}_{Set}$, para cada $a \in A$, $H(a)\colon \alpha_A(a) \to \alpha_A(a)$ debe ser morfismo en $\mathcal{C}$, es decir, debe pertenecer al conjunto $Morf_{\mathcal{C}}(\alpha_A(a), \alpha_A(a))$. Por tanto, si en los conjuntos anteriores existen más morfismos que la identidad, hay más de una posibilidad para definir la aplicación H.

Así, en general, $\mathbb{1}$ no es objeto final en $\mathcal{C}_{Set}$. Sin embargo, nos interesa una cierta subcategoría de $\mathcal{C}_{Set}$ en la que el objeto $\mathbb{1}$ es final, subcategoría que introducimos a continuación.

Si $\mathcal{C}$ es una categoría, denotamos por $\mathcal{C}^{\{\}}$ a la categoría que posee los mismos objetos que $\mathcal{C}$ y cuyos morfismos son, únicamente, las identidades. Según la terminología utilizada en Teoría de Categorías (ver en preliminares página 18, o en [74] página 11), $\mathcal{C}^{\{\}}$ es una *categoría discreta*.

Así, la categoría $\mathcal{C}^{\{\}}$ se reduce solamente a la clase de sus objetos. En el caso de que $\mathcal{C}$ sea una categoría pequeña, $\mathcal{C}^{\{\}}$ es solo un conjunto. Esto explica el uso de la notación $\{\}$, ya que parece que $\mathcal{C}^{\{\}}$ está vacía de significado como categoría. Sin embargo, nos va a servir para definir correctamente la categoría de familias de álgebras de $\mathcal{C}$, una subcategoría de $\mathcal{C}_{Set}$ que será importante en este trabajo para el caso particular en que $\mathcal{C}$ sea la categoría de álgebras para una signatura.

Usaremos la notación $\mathcal{C}_{Set}^{\{\}}$ para referirnos a $(\mathcal{C}^{\{\}})_{Set}$. Notar que no equivale a $(\mathcal{C}_{Set})^{\{\}}$, puesto que en $(\mathcal{C}^{\{\}})_{Set}$ hay morfismos que no son las identidades. Los objetos de $\mathcal{C}_{Set}^{\{\}}$ son los mismos que los de $\mathcal{C}_{Set}$, sin embargo, los morfismos se simplifican notablemente. Un morfismo entre dos objetos de $\mathcal{C}_{Set}^{\{\}}$, $\alpha_A\colon A \to Obj(\mathcal{C})$ y $\alpha_B\colon B \to Obj(\mathcal{C})$, es solo una aplicación $h\colon A \to B$ que hace conmutativo el diagrama



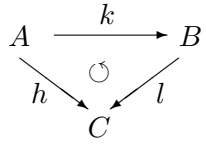
Podemos identificar $\mathcal{C}_{Set}^{\{\}}$ con la subcategoría de $\mathcal{C}_{Set}$ que tiene los mismos objetos que $\mathcal{C}_{Set}$ y cuyos morfismos son las parejas (h, H) tal que h hace conmutativo el diagrama anterior y $H(a)$ siempre es la aplicación identidad. De hecho, de ahora en adelante, pensaremos en $\mathcal{C}_{Set}^{\{\}}$ como subcategoría de $\mathcal{C}_{Set}$.

Observar que para poder ver $\mathcal{C}_{Set}^{\{\}}$ como subcategoría de $\mathcal{C}_{Set}$, la condición de conmutatividad del diagrama anterior no es suficiente ya que no caracteriza a los morfismos de $\mathcal{C}_{Set}$ que están en $\mathcal{C}_{Set}^{\{\}}$, debido a que la condición de conmutatividad fija la aplicación h pero el grado de libertad aportado por los conjuntos $Morf_{\mathcal{C}}(\alpha_A(a), \alpha_A(a))$ no determina H, por lo que hemos tenido que fijarla.

Teniendo en cuenta este último comentario, la proposición 2.3.4 tiene una consecuencia evidente. Dado $\alpha_A\colon A \to Obj(\mathcal{C})$ objeto de la categoría $\mathcal{C}_{Set}^{\{\}}$, el morfismo $(\alpha_A, id_{\alpha_A})\colon \alpha_A \to \mathbb{1}$ de $\mathcal{C}_{Set}$ utilizado para probar la existencia de morfismos desde cualquier objeto de $\mathcal{C}_{Set}$ en el objeto $\mathbb{1}$ es también morfismo de $\mathcal{C}_{Set}^{\{\}}$ (para cada $a \in A$, $H(a)$ es la identidad de $\alpha_A(a)$). Además, $(\alpha_A, id_{\alpha_A})$ es el único morfismo en $\mathcal{C}_{Set}^{\{\}}$ de α_A en $\mathbb{1}$ ya que cualquier otro morfismo $(g, G)\colon \alpha_A \to \mathbb{1}$ de $\mathcal{C}_{Set}^{\{\}}$ debe verificar que $g = \alpha_A$, y, por tanto, la unicidad está garantizada. Como consecuencia se tiene el siguiente corolario.

Corolario 2.3.5 *El objeto $\mathbb{1}$ es final en la categoría $\mathcal{C}_{Set}^{\{\}}$.*

En general, dada una categoría $\mathcal{D}$ y fijado C objeto de $\mathcal{D}$, se define la *categoría "slice" $\mathcal{D}$ sobre C*, denotada por $\mathcal{D}/C$ (ver en preliminares página 21, en [10] o en [79], entre otros), como la categoría que tiene por objetos los morfismos $h\colon A \to C$ de $\mathcal{D}$ y, por morfismos, los de $\mathcal{D}$ que hacen el triángulo correspondiente conmutativo. Es decir, si $h\colon A \to C$ y $l\colon B \to C$ son objetos de $\mathcal{D}/C$, los morfismos entre ellos son morfismos $k\colon A \to B$ en $\mathcal{D}$ tales que $l \circ k = h$.



Es evidente, tal y como se afirma en [79], que $id_C \colon C \to C$ es objeto final en $\mathcal{D}/C$.

Cuando $\mathcal{C}$ es una categoría pequeña, $Obj(\mathcal{C})$ es un objeto de la categoría de conjuntos y podemos considerar la categoría $Set/Obj(\mathcal{C})$. Así, $Set/Obj(\mathcal{C})$ tiene por objetos a las aplicaciones $\alpha \colon A \to Obj(\mathcal{C})$ y por morfismos entre dos objetos, $\alpha \colon A \to Obj(\mathcal{C})$ y $\beta \colon B \to Obj(\mathcal{C})$, las aplicaciones $f \colon A \to B$ que verifican $\beta \circ f = \alpha$. Es decir, nuestra construcción $\mathcal{C}_{Set}^{\{\}}$ coincide con la categoría $Set/Obj(\mathcal{C})$, coincidiendo también, obviamente, la descripción de su objeto final. Hemos preferido, sin embargo, mostrar el proceso general de construcción a partir de la categoría $\mathcal{C}_{Set}$ ya que este mismo proceso tendrá sentido cuando nos ocupemos, en el siguiente capítulo, de categorías de implementaciones. La tabla 2.1 recoge las construcciones introducidas sobre una categoría $\mathcal{C}$.

Notación	**Descripción**	Objeto final	Coproducto
$\mathcal{C}^{\{\}}$	*Obj.:* los de $\mathcal{C}$ *Morf.:* las identidades	No	No
$\mathcal{C}_{Set}$	*Obj.:* familias indexadas de objetos de $\mathcal{C}$ *Morf.:* pares de aplicaciones, una entre los conjuntos de índices y otra que asocia a cada índice un morfismo de $\mathcal{C}$	lo tiene cuando $\mathcal{C}$ tiene objeto final	Sí
$\mathcal{C}_{Set}^{\{\}}$	*Obj.:* los de $\mathcal{C}_{Set}$ *Morf.:* aplicaciones entre los conjuntos de índices	Sí ($\mathbb{1}$)	Sí

Tabla 2.1: Operaciones definidas sobre una categoría $\mathcal{C}$

2.4 Interpretación formal de las Σ_{imp}-álgebras como familias de Σ-álgebras

Dada una signatura Σ, nos interesa estudiar la categoría $Alg(\Sigma_{imp})$. Disponer de la operación $()_{Set}$, nos permite formalizar la interpretación dada anteriormente para las Σ_{imp}-álgebras como familias de Σ-álgebras. Para ello, definimos el funtor:

$$I^{imp} \colon Alg(\Sigma_{imp}) \to Alg(\Sigma)_{Set}$$

Si $A = \langle A_{imp_\Sigma}, (A_g)_{g\in G}, \{imp\_\sigma_A \colon A_{imp_\Sigma} \times A_\omega \to A_v\}_{(\sigma\colon \omega\to v)\in\Sigma}\rangle$ es una Σ_{imp}-álgebra, $I^{imp}(A)$ es la aplicación $\alpha_{A_{imp_\Sigma}} \colon A_{imp_\Sigma} \to Obj(Alg(\Sigma))$ que a cada $a \in A_{imp_\Sigma}$ le hace corresponder la Σ-álgebra $\alpha_{A_{imp_\Sigma}}(a) := \langle (A_g)_{g\in G}, \{imp\_\sigma_A(a,-) \colon A_\omega \to A_v\}_{(\sigma\colon \omega\to v)\in\Sigma}\rangle$. Respecto de los morfismos, si $A = \langle A_{imp_\Sigma}, (A_g)_{g\in G}, \{imp\_\sigma_A \colon A_{imp_\Sigma} \times A_\omega \to A_v\}_{(\sigma\colon \omega\to v)\in\Sigma}\rangle$ y $B = \langle B_{imp_\Sigma}, (B_g)_{g\in G}, \{imp\_\sigma_B \colon B_{imp_\Sigma} \times B_\omega \to B_v\}_{(\sigma\colon \omega\to v)\in\Sigma}\rangle$ son Σ_{imp}-álgebras y $f = (f_{imp_\Sigma}, (f_g)_{g\in G})$ es Σ_{imp}-morfismo de A en B, se define $I^{imp}(f) \colon I^{imp}(A) \to I^{imp}(B)$ como el morfismo $(f_{imp_\Sigma}, (f_g)_{g\in G})$ en $Alg(\Sigma)_{Set}$.

I^{imp} está bien definido ya que $I^{imp}(f)$ es realmente un morfismo de $Alg(\Sigma)_{Set}$ por ser $(f_g)_{g\in G}$ un morfismo en $Alg(\Sigma)$ entre $\alpha_{A_{imp_\Sigma}}(a)$ y $\alpha_{B_{imp_\Sigma}}(f_{imp_\Sigma}(a))$.

Vamos a señalar algunas de las propiedades del funtor I^{imp}.

1. Tal y como se ha definido el funtor I^{imp}, es evidente que es fiel e inyectivo sobre objetos.

2. La imagen de I^{imp} no cubre todos los objetos de $Alg(\Sigma)_{Set}$. Los objetos de $Alg(\Sigma)_{Set}$ que están en la imagen de I^{imp} verifican que todas las Σ-álgebras de la familia poseen los mismos soportes para los géneros de G.

   Por ejemplo, tomamos la signatura $\Sigma = \langle\{g\}, \{\sigma\colon g \to g\}\rangle$ y definimos el siguiente objeto de $Alg(\Sigma)_{Set}$: $\alpha_{\{1,2\}}\colon \{1,2\} \to Obj(Alg(\Sigma))$ con $\alpha_{\{1,2\}}(1) := \langle\mathbb{Z}, \{id\}\rangle$ y $\alpha_{\{1,2\}}(2) := \langle\mathbb{N}, \{id\}\rangle$. Es evidente que $\alpha_{\{1,2\}}$ no está en la imagen del funtor I^{imp} ya que las Σ-álgebras que define poseen distintos soportes para el género g.

3. I^{imp} tampoco es pleno. De la definición de I^{imp} se desprende que si un morfismo (h, H) de $Alg(\Sigma)_{Set}$ es la imagen por I^{imp} de un morfismo de $Alg(\Sigma_{imp})$, entonces la aplicación H es constante, en el sentido de que $H(a)$ es siempre el mismo Σ-morfismo $\big((f_g)_{g\in G}\big)$ para cualquier índice a. Es claro que no todos los $Alg(\Sigma)_{Set}$-morfismos son de este tipo.

2.4.1 El objeto $\mathbb{1}$ en $Alg(\Sigma)_{Set}$

Ya hemos visto que si $\mathcal{C}$ es un categoría pequeña, entonces existe en $\mathcal{C}_{Set}$ un objeto, al que hemos denotado por $\mathbb{1}$, y que es objeto final en la subcategoría $\mathcal{C}^{\{\}}_{Set}$. Vamos a estudiar la relación entre el objeto $\mathbb{1}$ y la categoría $Alg(\Sigma_{imp})$.

La primera observación es que $\mathbb{1}$ no es un objeto de la categoría $Alg(\Sigma)_{Set}$ puesto que la categoría $Alg(\Sigma)$ no es una categoría pequeña. Ahora bien, si fijamos un conjunto $\mathcal{U}$ e imponemos que todos los soportes para las Σ-álgebras sean elegidos en $\wp(\mathcal{U})$ (conjunto de los subconjuntos de $\mathcal{U}$), la categoría $Alg(\Sigma)$ es pequeña. La única condición impuesta a $\mathcal{U}$ es que éste sea realmente un conjunto y no una clase propia.

Fijar un conjunto $\mathcal{U}$ es natural por el hecho de que nuestro interés real es trabajar con implementaciones, donde esto no supone restricción alguna. Al pasar a implementaciones, se utiliza un lenguaje de programación concreto y todos los datos que se manejan en las implementaciones deben formar parte del universo del lenguaje, por lo que realmente el equivalente en las implementaciones a los soportes de la parte algebraica, serán subconjuntos de ese universo. A partir de ahora supondremos un conjunto $\mathcal{U}$ fijo, que llamaremos *universo algebraico*, en el que se tomarán los soportes de las Σ-álgebras, y, nos referiremos por $Alg(\Sigma)$ a la categoría cuyas álgebras tienen soportes en el conjunto $\mathcal{U}$.

En estas condiciones, $\mathbb{1}$ es un objeto de $Alg(\Sigma)_{Set}$.

Volviendo sobre la inclusión $I^{imp}\colon Alg(\Sigma_{imp}) \to Alg(\Sigma)_{Set}$, es evidente que el objeto $\mathbb{1}$ no está en su imagen (basta observar que los soportes de las Σ-álgebras de la imagen de la aplicación $\mathbb{1}$ son distintos). Como consecuencia, la subcategoría de $Alg(\Sigma)_{Set}$ imagen por la inclusión I^{imp} no es cerrada por el coproducto heredado de $Alg(\Sigma)_{Set}$.

2.4.2 Subcategorías de álgebras con soportes fijos

Vamos a descomponer la categoría $Alg(\Sigma_{imp})$ en subcategorías. Agruparemos en una misma subcategoría las Σ_{imp}-álgebras que tengan los mismos soportes para los géneros de la signatura de partida Σ y estudiaremos en estas subcategorías la existencia de coproducto y de objeto final. No siempre queda claro en Matemáticas que una estructura algebraica está constituida por dos piezas: un conjunto (o varios) y unas operaciones. Estamos acostumbrados a identificar la estructura con solo una de sus componentes, lo que en terminología del Álgebra Universal conocemos por soporte. Así, por ejemplo, son muchos los grupos que comparten como soporte el conjunto de los números enteros. A la hora de programar, esto es importante ya que el soporte corresponde con lo que serán los elementos de la estructura, y tener el mismo soporte es algo así como tener el mismo tipo de elementos.

Si G es el conjunto de géneros de Σ, para cada G-conjunto $D = (D_g)_{g\in G}$ sobre $\mathcal{U}$, denotamos por $Alg^D(\Sigma)$ a la subcategoría plena de $Alg(\Sigma)$ cuyos objetos son las Σ-álgebras de $Alg(\Sigma)$ que tienen por soporte, para cada género g de G, el conjunto D_g. $Alg^D(\Sigma)_{Set}$ denota a la categoría obtenida aplicando la operación $()_{Set}$ a la categoría $Alg^D(\Sigma)$.

Por otra parte, denotamos por $Alg^D(\Sigma_{imp})$ a la subcategoría plena de $Alg(\Sigma_{imp})$ que tiene por objetos a las Σ_{imp}-álgebras cuyos soportes para los géneros de G son los conjuntos de D. Esta notación nos puede conducir a engaño ya que no es del todo coherente. Para cada G-conjunto D, la construcción $Alg^D(\Sigma)$ es distinta de la construcción $Alg^D(\Sigma_{imp})$, la primera fija todos los soportes de las Σ-álgebras, la segunda deja libre el correspondiente al género imp_Σ.

Cada categoría $Alg^D(\Sigma)$ es pequeña (incluso eliminando la restricción respecto al universo algebraico $\mathcal{U}$), por lo que en $Alg^D(\Sigma)_{Set}$, existe objeto distinguido, que denotaremos por $\mathbb{1}^D\colon Obj(Alg^D(\Sigma)) \to Obj(Alg^D(\Sigma))$.

Es fácil observar que la imagen de la restricción del funtor I^{imp} a la subcategoría $Alg^D(\Sigma_{imp})$ cae dentro de $Alg^D(\Sigma)_{Set}$. Abusando de la notación, escribiremos también $I^{imp}\colon Alg^D(\Sigma_{imp}) \to Alg^D(\Sigma)_{Set}$. Además, ambas categorías poseen coproductos arbitrarios y el funtor I^{imp} los conserva.

La siguiente proposición recoge un primer resultado que relaciona las subcategorías $Alg^D(\Sigma_{imp})$ y $Alg^D(\Sigma)_{Set}$.

Proposición 2.4.1 *Para cada G-conjunto D, el funtor I^{imp} induce una biyección entre los objetos de las categorías $Alg^D(\Sigma_{imp})$ y $Alg^D(\Sigma)_{Set}$.*

La biyección existente entre $Obj(Alg^D(\Sigma_{imp}))$ y $Obj(Alg^D(\Sigma)_{Set})$ nos permite considerar un objeto distinguido en la categoría $Alg^D(\Sigma_{imp})$, el trasladado del objeto $\mathbb{1}^D$ por la biyección inducida por I^{imp}, que puede describirse por

$$\Big\langle Obj(Alg^D(\Sigma)), D, \{imp\_\sigma_{\mathbb{1}^D}\colon Obj(Alg^D(\Sigma)) \times D_\omega \to D_v\}_{(\sigma\colon \omega\to v)\in\Sigma}\Big\rangle$$

estando las operaciones definidas por $imp\_\sigma_{\mathbb{1}^D}(A, d_\omega) := \sigma_A(d_\omega)$, para cada $A = \big\langle D, \{\sigma_A\colon D_\omega \to D_v\}_{(\sigma\colon \omega\to v)\in\Sigma}\big\rangle$ objeto de $Alg^D(\Sigma)$ y para todo $d_\omega \in D_\omega$. Abusando en la notación, si no da lugar a confusión, nos referiremos también a este objeto por $\mathbb{1}^D$.

Ejemplo 2.4.2 Consideramos la signatura `GRP` introducida en el ejemplo 2.2.1, y sea D cualquier conjunto. La expresión del objeto final $\mathbb{1}^D_{\mathtt{GRP}_{imp}}$, en este caso es la siguiente:

- El soporte para el género distinguido $imp_{\mathtt{GRP}}$ es el conjunto $Obj(Alg^D(\mathtt{GRP}))$, conjunto que podemos describir del siguiente modo:

$$Obj(Alg^D(\mathtt{GRP})) = \big\{\langle D, f\rangle \;\mid\; f = (f_{prd}\colon D \times D \to D, f_{inv}\colon D \to D, f_{unt}\colon \{*\} \to D)\big\},$$

- las operaciones vienen definidas por:
  - $imp\_prd_{\mathbb{1}^D_{\mathtt{GRP}_{imp}}} : Obj(Alg^D(\mathtt{GRP})) \times D \times D \to D$, dada, para cada $\langle D, f\rangle \in Obj(Alg^D(\mathtt{GRP}))$ y para cada $d_1, d_2 \in D$, por $imp\_prd_{\mathbb{1}^D_{\mathtt{GRP}_{imp}}}(\langle D, f\rangle, d_1, d_2) := f_{prd}(d_1, d_2)$;
  - $imp\_inv_{\mathbb{1}^D_{\mathtt{GRP}_{imp}}} : Obj(Alg^D(\mathtt{GRP})) \times D \to D$ definida por $imp\_inv_{\mathbb{1}^D_{\mathtt{GRP}_{imp}}}(\langle D, f\rangle, d_1) := f_{inv}(d_1)$, para cada $\langle D, f\rangle \in Obj(Alg^D(\mathtt{GRP}))$ y para cada $d_1 \in D$; e,
  - $imp\_unt_{\mathbb{1}^D_{\mathtt{GRP}_{imp}}} : Obj(Alg^D(\mathtt{GRP})) \to D$ definida, para cada $\langle D, f\rangle \in Obj(Alg^D(\mathtt{GRP}))$, por $imp\_unt_{\mathbb{1}^D_{\mathtt{GRP}_{imp}}}(\langle D, f\rangle) := f_{unt}(*)$.

□

Por otra parte, el conjunto de Σ-álgebras de $Alg^D(\Sigma)$ es biyectivo con el espacio de funciones $\prod_{(\sigma\colon \omega \to v) \in \Sigma} D_v^{D_\omega}$ (como es habitual, si X e Y son conjuntos Y^X denota al conjunto de funciones de X en Y). Así, se tiene otra expresión para el objeto $\mathbb{1}^D$ de $Alg^D(\Sigma_{imp})$:

$$\Big\langle \prod_{(\sigma\colon \omega' \to v') \in \Sigma} D_{v'}^{D_{\omega'}}, D, \{imp\_\sigma_{\mathbb{1}^D}\colon \prod_{(\sigma\colon \omega' \to v') \in \Sigma} D_{v'}^{D_{\omega'}} \times D_\omega \to D_v\}_{(\sigma\colon \omega' \to v') \in \Sigma} \Big\rangle$$

siendo $imp\_\sigma_{\mathbb{1}^D}\big((f_\delta)_{(\delta\colon \omega' \to v') \in \Sigma}, d_\omega\big) := f_\sigma(d_\omega)$, para todo $(f_\delta)_{(\delta\colon \omega' \to v') \in \Sigma} \in \prod_{(\sigma\colon \omega' \to v') \in \Sigma} D_{v'}^{D_{\omega'}}$ y para todo $d_\omega \in D_\omega$.

Ejemplo 2.4.3 Aplicando lo anterior al ejemplo de la signatura $\mathtt{GRP}_{imp}$, se tiene que el conjunto $Obj(Alg^D(\mathtt{GRP}))$ es biyectivo con el espacio de funciones $D^{D\times D} \times D^D \times D$. Podemos considerar este espacio de funciones como soporte para el género $imp_{\mathtt{GRP}}$, obteniendo así otra descripción de la $\mathtt{GRP}_{imp}$-álgebra final. La descripción completa es la siguiente:

- El soporte para el género distinguido $imp_{\mathtt{GRP}}$ es el conjunto $D^{D\times D} \times D^D \times D$
- las operaciones vienen definidas por:
  - $imp\_prd_{\mathbb{1}^D_{\mathtt{GRP}_{imp}}} : (D^{D\times D} \times D^D \times D) \times D \times D \to D$, dada, para cada $(f_{prd}, f_{inv}, d_0) \in D^{D\times D} \times D^D \times D$ y para cada $d_1, d_2 \in D$, por $imp\_prd_{\mathbb{1}^D_{\mathtt{GRP}_{imp}}}\big((f_{prd}, f_{inv}, d_0), d_1, d_2\big) := f_{prd}(d_1, d_2)$;
  - $imp\_inv_{\mathbb{1}^D_{\mathtt{GRP}_{imp}}} : (D^{D\times D} \times D^D \times D) \times D \to D$, definida como $imp\_inv_{\mathbb{1}^D_{\mathtt{GRP}_{imp}}}\big((f_{prd}, f_{inv}, d_0), d_1\big) := f_{inv}(d_1)$, para todo $(f_{prd}, f_{inv}, d_0) \in D^{D\times D} \times D^D \times D$ y para todo $d_1 \in D$;
  - $imp\_unt_{\mathbb{1}^D_{\mathtt{GRP}_{imp}}} : (D^{D\times D} \times D^D \times D) \to D$, dada por $imp\_unt_{\mathbb{1}^D_{\mathtt{GRP}_{imp}}}\big((f_{prd}, f_{inv}, d_0)\big) := d_0$, para todo $(f_{prd}, f_{inv}, d_0) \in D^{D\times D} \times D^D \times D$.

□

Esta última expresión del objeto $\mathbb{1}^D$ es más cercana a las implementaciones. Cada dato del soporte del género distinguido es una tupla de funciones por lo que, si pretendemos trasladar directamente esta idea a implementaciones, resulta bastante natural el uso de la programación funcional.

La proposición 2.4.1 muestra la existencia de una biyección entre los objetos de las categorías $Alg^D(\Sigma_{imp})$ y $Alg^D(\Sigma)_{Set}$. Sin embargo, no ocurre lo mismo entre los morfismos, puesto que la categoría $Alg^D(\Sigma)_{Set}$ posee más morfismos que $Alg^D(\Sigma_{imp})$. Por tanto, la imagen $Im\ I^{imp}$ no es una subcategoría plena. Los morfismos de esta subcategoría imagen pueden caracterizarse fácilmente. Por definición de la inclusión I^{imp}, es evidente que todos los morfismos de la imagen son de la forma $(h, (f_g)_{g\in G})$, es decir, la aplicación que asocia a cada índice un morfismo, es constante. Pero además, es sencillo comprobar que también se tiene el recíproco, resultado que recogemos en la siguiente proposición.

Proposición 2.4.4 *$Im\ I^{imp}$ es la subcategoría de $Alg^D(\Sigma)_{Set}$ cuyos morfismos son los pares (h, H) tales que H es constante.*

Con ello, tenemos caracterizadas las familias de Σ-álgebras que pueden interpretarse como Σ_{imp}-álgebras.

2.4.3 El objeto $\mathbb{1}^D$

En esta sección vamos a particularizar a $Alg^D(\Sigma)_{Set}$ las nociones estudiadas anteriormente para las categorías obtenidas por la operación $()_{Set}$. Así, estudiaremos cómo $\mathbb{1}^D$ es final en una adecuada subcategoría de $Alg^D(\Sigma)_{Set}$.

Utilizando la misma notación que en el caso general, denotamos por $Alg^{\{\}}(\Sigma)$ a la subcategoría de $Alg(\Sigma)$ cuyos morfismos son únicamente las identidades, por $Alg^{\{\}}(\Sigma_{imp})$ a la subcategoría de $Alg(\Sigma_{imp})$ cuyos morfismos son la identidad sobre los soportes de los géneros de la signatura Σ y por $Alg^{\{\}}(\Sigma)_{Set}$ a la subcategoría de $Alg(\Sigma)_{Set}$ obtenida a partir de la categoría $Alg^{\{\}}(\Sigma)$.

Análogamente, fijado un G-conjunto D, denotamos por $Alg^{D,\{\}}(\Sigma)$ a la subcategoría de $Alg^D(\Sigma)$ cuyos únicos morfismos son las identidades, $Alg^{D,\{\}}(\Sigma_{imp})$ denotará a la subcategoría de $Alg^D(\Sigma_{imp})$ cuyos morfismos son los Σ_{imp}-morfismos que son identidades sobre el G-conjunto D. Finalmente, denotamos por $Alg^{D,\{\}}(\Sigma)_{Set}$ a la subcategoría de $Alg^D(\Sigma)_{Set}$ obtenida a partir de la categoría $Alg^{D,\{\}}(\Sigma)$ mediante la operación $()_{Set}$.

La tabla 2.2 recoge la relación entre todas estas categorías.

Un primer resultado que se obtiene a partir de la proposición 2.4.4, es que las categorías $Alg^{D,\{\}}(\Sigma_{imp})$ y $Alg^{D,\{\}}(\Sigma)_{Set}$ son isomorfas.

Por su parte, el resultado análogo al teorema 2.3.3, que caracteriza al objeto $\mathbb{1}$ de cualquier categoría del tipo $\mathcal{C}_{Set}$ como coproducto de una familia de objetos, se escribe en los siguientes términos:

el objeto $\mathbb{1}^D\colon Obj(Alg^D(\Sigma)) \to Obj(Alg^D(\Sigma))$ de $Alg^{D,\{\}}(\Sigma)_{Set}$ es el coproducto en dicha categoría de los objetos de la imagen de la inclusión canónica de $Alg^{D,\{\}}(\Sigma)$ en $Alg^{D,\{\}}(\Sigma)_{Set}$.

Para interpretar el resultado anterior en la categoría $Alg^D(\Sigma_{imp})$ consideramos la inclusión canónica de $Alg^D(\Sigma)$ en $Alg^D(\Sigma_{imp})$ (aunque no lo hemos citado hasta el momento, es fácil observar que hay un modo natural de ver una Σ-álgebra como Σ_{imp}-álgebra, basta tomar el

	Categoría	**Fijado D como G-conjunto**	**Morfismos que sobre los soportes de Σ son la identidad**	**Fijado D como G-conjunto y con morfismos identidad sobre D**
Operación	$Alg(\Sigma)$	$Alg^D(\Sigma)$	$Alg^{\{\}}(\Sigma)$	$Alg^{D,\{\}}(\Sigma)$
$()_{Set}$	$Alg(\Sigma)_{Set}$	$Alg^D(\Sigma)_{Set}$	$Alg^{\{\}}(\Sigma)_{Set}$	$Alg^{D,\{\}}(\Sigma)_{Set}$
$()_{imp}$	$Alg(\Sigma_{imp})$	$Alg^D(\Sigma_{imp})$	$Alg^{\{\}}(\Sigma_{imp})$	$Alg^{D,\{\}}(\Sigma_{imp})$

Tabla 2.2: Categorías introducidas mediante las operaciones $()_{Set}$ e $()_{imp}$

conjunto unipuntual como soporte para el nuevo género imp_Σ) y su restricción a $Alg^{D,\{\}}(\Sigma)$, que obviamente cae dentro de $Alg^{D,\{\}}(\Sigma_{imp})$. Se tiene entonces el siguiente resultado que corresponde al teorema 2.3.3 para el caso de Σ_{imp}-álgebras:

Teorema 2.4.5 *El objeto $\mathbb{1}^D$ es el coproducto en $Alg^{D,\{\}}(\Sigma_{imp})$ de los objetos de la imagen de la inclusión canónica de $Alg^D(\Sigma)$ en $Alg^D(\Sigma_{imp})$.*

$$\begin{array}{ccc} Alg^{D,\{\}}(\Sigma) & \longrightarrow & Alg^{D,\{\}}(\Sigma_{imp}) \cong Alg^{D,\{\}}(\Sigma)_{Set} \\ \downarrow & \circlearrowleft & \downarrow \\ Alg^D(\Sigma) & \longrightarrow & Alg^D(\Sigma_{imp}) \end{array}$$

Se tienen además los siguientes resultados de finalidad:

Teorema 2.4.6 *El objeto $\mathbb{1}^D$ es final en la categoría $Alg^{D,\{\}}(\Sigma_{imp})$.*

El resultado de finalidad anterior junto con el hecho de que $Obj(Alg^{\{\}}(\Sigma_{imp})) = \bigsqcup_{D\in\wp(\mathcal{U})^G} Obj(Alg^{D,\{\}}(\Sigma_{imp}))$, nos permiten caracterizar los objetos $\mathbb{1}^D$ por una propiedad universal que explica cómo se *agrupan* todos esos objetos finales.

Teorema 2.4.7 *La familia $\{\mathbb{1}^D\}_{D\in\wp(\mathcal{U})^G}$ es final en la categoría $Alg^{\{\}}(\Sigma_{imp})$.*

2.5 Interpretación en términos de especificaciones ocultas

En las secciones anteriores nos hemos ceñido al marco de especificación algebraica clásica, basado en el concepto matemático de álgebra para una signatura. El uso de esta teoría en trabajos

dedicados a la formalización de tipos de datos es muy frecuente, pero la gran diversidad de estructuras de datos que aparecen en el ámbito de la programación ha motivado la aparición de diferentes variantes sobre el concepto de álgebra para una signatura. Una de estas variantes es la noción de álgebra oculta, que es la base de lo que se conoce como *Especificación Oculta*. Vamos en primer lugar a dar algunas ideas básicas sobre este campo y posteriormente estableceremos conexiones con las construcciones presentadas en las secciones anteriores.

2.5.1 Especificaciones ocultas

La teoría de especificaciones ocultas tiene su origen en el trabajo [43], desarrollado por Goguen en 1991, si bien la primera exposición sistemática y formal fue dada en [44]. Desde entonces ha sido considerada un área de gran interés, dando lugar a numerosos trabajos. Algunos de ellos son [46], [23], [45], [27], [76], [94] y [47].

Los conceptos básicos son los de signatura y álgebra ocultas. Aunque las definiciones varían ligeramente de unos trabajos a otros, la idea fundamental es distinguir entre dos clases de géneros: géneros ocultos y géneros visibles. Su interpretación desde el paradigma orientado a objeto es que los géneros ocultos corresponderán con los conjuntos de objetos mientras que los visibles lo harán con los de datos.

En el marco oculto se exige fijar una representación común para los datos que se van a manipular. Desde el punto de vista algebraico esto se traduce en fijar un álgebra para lo que será la parte visible de la especificación.

Así, fijamos un álgebra $(V\Sigma, D)$, que denominaremos *álgebra de datos*, donde $V\Sigma = \langle VG, V\Omega\rangle$ es una signatura, que llamaremos *signatura de datos*, y D es una $V\Sigma$-álgebra verificando que, para todo $v \in VG$, el soporte D_v no es vacío. A los elementos de VG se les denomina *géneros visibles* y a los de $V\Omega$ *operaciones visibles*. Además, incluimos en $V\Omega$, como constantes, los elementos de los soportes de la $V\Sigma$-álgebra D que no existan como constantes en $V\Omega$. Esta condición se incluye para asegurar que todos los datos visibles pueden ser generados, es decir, que el álgebra de datos generable contiene a la $V\Sigma$-álgebra D. Por tanto, si a partir de la signatura de datos se puede asegurar que todos los datos visibles pueden ser generados por las operaciones de la signatura, no será necesario incluir esta condición. El álgebra de datos $(V\Sigma, D)$ constituye una referencia fija para las observaciones que nos pueden permitir conocer (y diferenciar) "datos ocultos".

Definición 2.5.1 *Fijada un álgebra de datos* $(V\Sigma, D)$*, una* signatura oculta sobre $(V\Sigma, D)$ *es una signatura* $H\Sigma = \langle G, \Omega\rangle$ *tal que:*

- $G = HG \sqcup VG$*; los elementos de* HG *se denominan* géneros ocultos *de* $H\Sigma$*.*
- $\Omega = H\Omega \sqcup V\Omega$*, y para cada operación* $\sigma : g_1 \ \ldots \ g_n \to g$ *en* $H\Omega$ *se verifican las dos propiedades siguientes:*
  - *en* $g_1, \ldots, g_n, g$ *existe, al menos, un género oculto,*
  - *si en* $g_1, \ldots, g_n$ *existe un género oculto, es único. En este caso, asumimos por convenio que el género oculto aparece como primer argumento, es decir, es* g_1*.*

Si no ha lugar a confusión, hablaremos de signatura oculta sin hacer referencia al álgebra de datos. La definición anterior es la utilizada en trabajos como [76], [44], [23], [27], habiendo

ligeras variaciones respecto a la que puede encontrarse, por ejemplo, en [46] (aunque luego se usa la que hemos dado aquí). En [94] y [31] se generaliza para permitir argumentos ocultos múltiples, y se desarrolla una teoría para ese caso.

Observar que las únicas operaciones de $H\Sigma$ que involucran exclusivamente a géneros visibles son las operaciones visibles, las de $V\Omega$. Teniendo en cuenta que el fijar un álgebra de datos viene motivado por la necesidad de disponer de una representación para los datos, el hecho de haber incluido los elementos del álgebra D como constantes en la signatura, hace que considerando simplemente las operaciones constantes, tengamos un sistema de representantes para los datos. Esto hace que las operaciones visibles (no constantes) sean superfluas desde el punto de vista de la constructibilidad de los datos (el resultado devuelto al aplicar cualquiera de ellas es un elemento de D, ya incluido como constante en la signatura). En este sentido observar que es natural considerar, no el álgebra de términos de la signatura de datos, sino un cociente de ésta en el que se haya identificado cada término visible con la constante obtenida por su interpretación en el álgebra D. Por ello, consideraremos como términos visibles solamente las constantes.

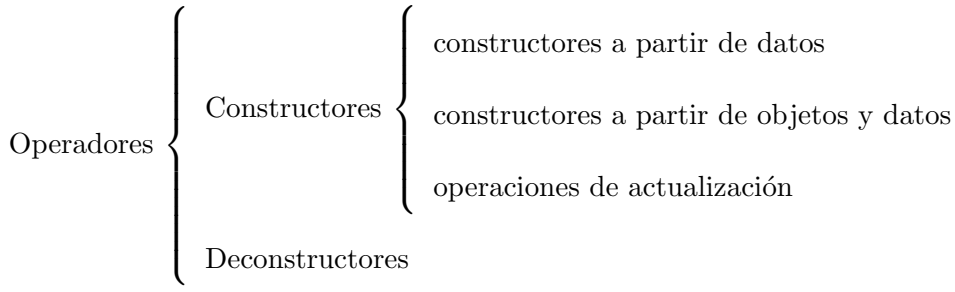
Sabiendo que una subsignatura de una signatura dada es una signatura cuyos conjuntos de géneros y de operaciones son subconjuntos de los de la signatura de partida (ver en preliminares, página 25), es claro que $V\Sigma$ es una subsignatura de $H\Sigma$.

En una signatura oculta se distinguen dos tipos de operaciones:

- operaciones que tienen como resultado un género oculto y que denominaremos *constructores*, y
- operaciones que terminan en un género visible y que llamaremos *deconstructores*.

Además los tipos de constructores quedan clasificados, dependiendo de los géneros de sus argumentos, del siguiente modo:

- constructores cuyos argumentos son todos géneros visibles, que denominaremos *constructores a partir de datos*. Para ellos usaremos la notación $\sigma\colon \omega \to h$, o escribiremos $\sigma \in H\Sigma_{\omega\ h}$, con $h \in HG$ y $\omega \in VG^*$;
- los que poseen un argumento de género oculto y distinto del género resultado que denominaremos *constructores a partir de objetos y datos*. Los denotaremos por $\sigma\colon h\ \omega \to h'$ o por $\sigma \in H\Sigma_{h\ \omega\ h'}$, con $h, h' \in HG,\ h \neq h'$ y $\omega \in VG^*$, y
- aquellos que tienen un argumento oculto y éste es el mismo que el género resultado; los denominaremos *operaciones de actualización* y los denotaremos por $\sigma\colon h\ \omega \to h$ o bien escribiremos $\sigma \in H\Sigma_{h\ \omega\ h}$, con $h \in HG$ y $\omega \in VG^*$.



Observar que, debido a la existencia de efectos laterales, en el último tipo de constructores se incluyen, junto a los que modifican un objeto de género oculto, aquellos que a partir de un objeto de género oculto construyen otro objeto del mismo género.

La terminología empleada habitualmente en los trabajos sobre especificaciones ocultas difiere de la nuestra. Como puede verse por ejemplo en [46], a los constructores a partir de datos se les llama *constantes ocultas generalizadas*, al resto de los constructores se les denomina *métodos* y a los deconstructores *atributos*. Sin embargo, el significado de los términos método y atributo no parece coincidir con el que se les da en los lenguajes de programación orientados a objeto, por lo que hemos preferido no adoptarla. Relacionando nuestra terminología con la empleada en la teoría clásica de especificación algebraica (por ejemplo en [40] y [72]), si consideramos los géneros ocultos como géneros distinguidos, los deconstructores se corresponden con los *observadores*. En relación con la terminología empleada por Cîrstea en [27] o por Reichel en [91], nuestros deconstructores se corresponden con sus *destructores* (término que hemos preferido evitar por considerarlo confuso en sí y por ser homónimo del que se emplea, por ejemplo en C++, para una operación de carácter totalmente distinto).

Introducimos a continuación dos ejemplos de signaturas ocultas que utilizaremos en el resto de la sección para ilustrar los conceptos que vayan apareciendo:

Ejemplo 2.5.2

1. Se trata de especificar un tipo de datos para modelar vectores en el semiplano superior (ordenada positiva). Para ello, identificamos cada vector con un par de números, un entero y un natural, que representan la proyección del vector en los ejes de abcisas y ordenadas respectivamente (incluyendo en la primera componente el signo). Consideramos operaciones que nos permitan:

   – conocer cada una de las componentes del vector,

   – saber si el valor de una componente del vector es un determinado número,

   – construir un vector a partir de otro vector y de los desplazamientos que queremos aplicar en ambos ejes,

   – construir un vector aplicando producto por un escalar a otro, y

   – construir el vector simétrico a uno dado.

   Para especificar el tipo de datos anterior, consideramos como signatura de datos $V\Sigma_1 = \langle \{int, nat, bool\}, \emptyset \rangle$ y como soportes para el álgebra de datos $D_{int} = \mathbb{Z}$, $D_{nat} = \mathbb{N}$ y $D_{bool} = \mathcal{B} = \{true, false\}$. Tomamos como signatura oculta $H\Sigma_1$ sobre $(V\Sigma_1, \{\mathbb{Z}, \mathbb{N}, \mathcal{B}\})$ la signatura

con un único género oculto, h, y cuyas operaciones son las siguientes:

$$\begin{array}{lcllll} abc & : & h & & \to & int \\ ord & : & h & & \to & nat \\ esx? & : & h & int & \to & bool \\ esy? & : & h & nat & \to & bool \\ move & : & h & int \quad nat & \to & h \\ esc & : & h & nat & \to & h \\ sim & : & h & & \to & h \end{array}$$

Las cuatro primeras operaciones son deconstructores, las tres últimas son operaciones de actualización.

2. Para especificar un tipo de datos que permita trabajar con grupos de enteros, consideramos el álgebra de datos $(V\Sigma_2, \mathbb{Z})$ formada por la signatura de datos $V\Sigma_2 = \langle\{v\}, \emptyset\rangle$ y tomando $\mathbb{Z}$ como soporte en el álgebra de datos para el único género visible v. Debemos considerar una signatura que, para cada grupo sobre $\mathbb{Z}$, permita calcular el producto del grupo, el inverso de cada elemento del grupo así como el elemento neutro. Para ello, consideramos la signatura oculta sobre $(V\Sigma_2, \mathbb{Z})$, que llamamos $H\Sigma_2$ formada por un género oculto h y cuyas operaciones son las siguientes:

$$\begin{array}{lcllll} prd\_grp & : & h \quad v \quad v & \to & v \\ inv\_grp & : & h \quad v & \to & v \\ unt\_grp & : & h & \to & v \end{array}$$

□

Un tipo particular de signaturas ocultas son aquellas cuyas operaciones son todas deconstructoras, signaturas que denominaremos *deconstructoras*, siguiendo la teminología introducida por Cîrstea en [27].

La signatura $H\Sigma_2$ del ejemplo anterior es deconstructora.

Introducimos la noción de morfismo entre signaturas ocultas.

Definición 2.5.3 *Sean $H\Sigma = \langle G, \Omega\rangle$ y $H\Sigma' = \langle G', \Omega'\rangle$ signaturas ocultas sobre $(V\Sigma, D)$. Un* morfismo entre signaturas ocultas *$\Phi\colon H\Sigma \to H\Sigma'$ es un par de aplicaciones $\Phi_{gen}\colon G \to G'$ y $\Phi_{ope}\colon \Omega \to \Omega'$ verificando las siguientes propiedades:*

i) Φ es morfismo de signaturas, es decir, para cada operación $\sigma\colon g_1 \dots g_n \to g$ en Ω, $\Phi_{ope}(\sigma)$ es una operación de Ω' con la siguiente aridad $\Phi_{ope}(\sigma)\colon \Phi_{gen}(g_1) \dots \Phi_{gen}(g_n) \to \Phi_{gen}(g)$;

ii) $\Phi_{gen}|_{VG} = id$;

iii) $\Phi_{ope}|_{V\Omega} = id$;

iv) $\Phi_{gen}(HG) \subseteq HG'$ *y,*

v) *si* $\sigma' \colon h'\omega \to g'$ *es operación en* $H\Sigma'$ *con* $h' \in \Phi_{gen}(HG)$*, entonces existe alguna operación* $\sigma \colon h\ \omega \to g$ *en* $H\Sigma$ *tal que* $\Phi_{ope}(\sigma) = \sigma'$.

Intuitivamente un morfismo entre signaturas ocultas es un morfismo entre signaturas definido como la identidad sobre la parte visible y que lleva la parte oculta a la oculta. Las signaturas ocultas con los morfismos entre ellas forman una categoría. De manera natural se tiene la noción de *isomorfismo entre signaturas ocultas* como un morfismo entre signaturas ocultas en el que las aplicaciones, entre los conjuntos de géneros y los de operaciones, son biyectivas.

Observando la definición de morfismo entre signaturas ocultas, es claro que se puede caracterizar la relación de isomorfismo entre signaturas ocultas: dos signaturas ocultas son isomorfas si y solo si son iguales bajo renombrado que respeta las partes visible y oculta.

Definición 2.5.4 *Un* álgebra oculta A para una signatura oculta $H\Sigma = \langle G, \Omega\rangle$ sobre $(V\Sigma, D)$ *es una* $H\Sigma$*-álgebra tal que* $A_{V\Sigma} = D$*, es decir, la restricción de* A *a* $V\Sigma$ *es el álgebra de datos* D.

Si no ha lugar a confusión hablaremos simplemente de álgebra para una signatura oculta.

Definición 2.5.5 *Si* $H\Sigma = \langle G, \Omega\rangle$ *es una signatura oculta sobre* $(V\Sigma, D)$ *y* A, B *son álgebras ocultas para* $H\Sigma$*, un* morfismo oculto entre A y B *es un* $H\Sigma$*-morfismo* $f \colon A \to B$ *tal que* $f_v = id_{D_v}$*, para todo* $v \in VG$.

Es obvio que, dada una signatura oculta $H\Sigma = \langle G, \Omega\rangle$ sobre $(V\Sigma, D)$, las $H\Sigma$-álgebras ocultas con los $H\Sigma$-morfismos ocultos constituyen una categoría, que denotaremos por $HAlg^D(H\Sigma)$.

A continuación mostramos algunas álgebras para las signaturas ocultas introducidas en el ejemplo 2.5.2.

Ejemplo 2.5.6

1. Definimos una $H\Sigma_1$-álgebra oculta A tomando como soporte para el género oculto h el conjunto $A_h = \{(x, y) \mid x \in \mathbb{Z},\ y \in \mathbb{N}\}$. Definimos las operaciones como sigue:

   · $abc_A\big((x, y)\big) = x$, para todo $(x, y) \in A_h$,

   · $ord_A\big((x, y)\big) = y$, para todo $(x, y) \in A_h$,

   · $esx?_A\big((x, y), z\big) = \begin{cases} true & \text{si } x = z \\ false & \text{en otro caso} \end{cases}$, para todo $(x, y) \in A_h$ y para todo $z \in \mathbb{Z}$,

   · $esy?_A\big((x, y), n\big) = \begin{cases} true & \text{si } y = n \\ false & \text{en otro caso} \end{cases}$, para todo $(x, y) \in A_h$ y para todo $n \in \mathbb{N}$,

- $move_A\big((x,y),z,n\big) = (x +_{\mathbb{Z}} z, y +_{\mathbb{N}} n)$, para todo $(x,y) \in A_h$, para todo $z \in \mathbb{Z}$ y para todo $n \in \mathbb{N}$,
- $esc_A\big((x,y),n\big) = (n *_{\mathbb{Z}} x, n *_{\mathbb{N}} y)$, para todo $(x,y) \in A_h$ y para todo $n \in \mathbb{N}$,
- $sim_A\big((x,y)\big) = (-_{\mathbb{Z}} x, y)$, para todo $(x,y) \in A_h$.

2. Definimos una $H\Sigma_2$-álgebra oculta que denominamos B tomando como soporte para el género h el conjunto $\mathbb{Z}$ y definiendo las operaciones del siguiente modo:

- $prd\_grp_B(z,z_1,z_2) = z_1 +_{\mathbb{Z}} z_2$, para todo $z,z_1,z_2 \in \mathbb{Z}$,
- $inv\_grp_B(z,z_1) = -_{\mathbb{Z}} z_1$, para todo $z,z_1 \in \mathbb{Z}$,
- $unt\_grp_B(z) = 0_{\mathbb{Z}}$, para todo $z \in \mathbb{Z}$.

□

Introducimos a continuación algunos conceptos que nos permitirán dar una expresión del objeto final en la categoría de álgebras ocultas.

Definición 2.5.7 *Sean $H\Sigma$ signatura oculta sobre $(V\Sigma, D)$ y h un género oculto de $H\Sigma$. Un $H\Sigma$-contexto para el género h de género v es un $H\Sigma$-término de género visible v con una única variable, que aparece una única vez y que es de género h.*

Denotaremos por $c[z_h]_v$ a un $H\Sigma$-contexto para el género h de género visible v con variable z_h. La expresión $\mathcal{C}_{H\Sigma}[z_h]_v$ denotará al conjunto de todos los $H\Sigma$-contextos para el género h de género v con variable z_h y, $\mathcal{C}_{H\Sigma}[z_h]$ al conjunto de todos los $H\Sigma$-contextos para el género h, de cualquier género visible y con variable z_h.

Por definición, $\mathcal{C}_{H\Sigma}[z_h] = \bigsqcup_{v \in VG} \mathcal{C}_{H\Sigma}[z_h]_v$, lo que nos permite interpretar $\mathcal{C}_{H\Sigma}[z_h]$ como un VG-conjunto $(\mathcal{C}_{H\Sigma}[z_h]_v)_{v \in VG}$.

Si no hay lugar a confusión nos referiremos a un $H\Sigma$-contexto para el género h de género v, por contexto para el género h, o simplemente por contexto (incluso omitiremos el nombre de la variable de género h ya que éste es irrelevante).

El hecho de que un contexto sea un término de género visible, se traduce en que es una expresión en la que la última operación aplicada es un deconstructor. Esta noción trata de recoger lo visible de un experimento que consta de algunos constructores seguidos de un deconstructor, que es el que proporciona la observación. Notar que, debido a que no trabajamos con el álgebra de términos de la signatura de datos, sino con un cociente suyo en el que podemos tomar una familia de representantes que son constantes, podemos suponer que los únicos términos visibles que pueden aparecer como subtérminos de un contexto son constantes.*

Veamos cómo son los contextos para las signaturas ocultas introducidas anteriormente.

Ejemplo 2.5.8

1. Volviendo a la signatura $H\Sigma_1$ del ejemplo 2.5.2, el conjunto de contextos para el género oculto h con variable z_h, puede descomponerse en tres conjuntos, ya que para cada uno de los tres géneros visibles de la signatura existe algún deconstructor que lo tiene como género resultado. Por tanto, $\mathcal{C}_{H\Sigma_1}[z_h] = \mathcal{C}_{H\Sigma_1}[z_h]_{int} \cup \mathcal{C}_{H\Sigma_1}[z_h]_{nat} \cup \mathcal{C}_{H\Sigma_1}[z_h]_{bool}$.

*El no haber prestado suficiente atención a este hecho llevó a Goguen y Malcolm a publicar en [46] una demostración incorrecta sobre la existencia de un objeto final oculto.

- $\mathcal{C}_{H\Sigma_1}[z_h]_{int}$:
  los contextos de género *int* se obtienen aplicando en último lugar el deconstructor *abc* que es el único cuyo género resultado es *int*. Así, $\mathcal{C}_{H\Sigma_1}[z_h]_{int}$ está formado por los contextos:
  - $abc(z_h)$
  - $abc(met_n(\dots(met_1(z_h, d_{\omega_1}), \dots), d_{\omega_n}))$ siendo, para cada $i \in \{1, \dots n\}$, $met_i \in \{move, esc, sim\}$ y

$$d_{\omega_i} \in \begin{cases} \mathbb{Z} \times \mathbb{N} & \text{si } met_i = move \\ \mathbb{N} & \text{si } met_i = esc \\ \text{nada} & \text{en otro caso } (met_i = sim) \end{cases}$$

- $\mathcal{C}_{H\Sigma_1}[z_h]_{nat}$:
  los de género *nat* se obtienen aplicando en último lugar *ord*, el único deconstructor de género *nat*. Así, tenemos como contextos de género *nat*:
  - $ord(z_h)$
  - $ord(met_n(\dots(met_1(z_h, d_{\omega_1}), \dots), d_{\omega_n}))$, donde para cada $i \in \{1, \dots n\}$, $met_i \in \{move, esc, sim\}$ y d_{ω_i} igual que en el caso anterior.

- $\mathcal{C}_{H\Sigma_1}[z_h]_{bool}$:
  los de género *bool* se obtienen aplicando en último lugar un deconstructor de género *bool*. En este caso, como hay dos deconstructores de género *bool*, *esx*? y *esy*?, hay más posibilidades para formar contextos:
  - $esx?(z_h, d_z)$ con $d_z \in \mathbb{Z}$,
  - $esy?(z_h, d_n)$ con $d_n \in \mathbb{N}$,
  - $esx?(met_n(\dots(met_1(z_h, d_{\omega_1}), \dots), d_{\omega_n}), d_z)$ con $d_z \in \mathbb{Z}$, y para cada $i \in \{1, \dots n\}$, $met_i \in \{move, esc, sim\}$ y d_{ω_i} igual que en los casos anteriores.
  - $esy?(met_n(\dots(met_1(z_h, d_{\omega_1}), \dots), d_{\omega_n}), d_n)$ con $d_n \in \mathbb{N}$, y para cada $i \in \{1, \dots n\}$, $met_i \in \{move, esc, sim\}$ y d_{ω_i} igual que en los casos anteriores.

2. Para la signatura $H\Sigma_2$ del ejemplo 2.5.2, por ser deconstructora, los contextos para el género oculto h con variable z_h tienen expresión mucho más sencilla. Así, $\mathcal{C}_{H\Sigma_2}[z_h]$ está formado por los siguientes contextos:
   - $prd\_grp(z_h, z_1, z_2)$ con $z_1, z_2 \in \mathbb{Z}$,
   - $inv\_grp(z_h, z_1)$ con $z_1 \in \mathbb{Z}$,
   - $unt\_grp(z_h)$.

□

Es fácil probar que lo que ocurre en el último ejemplo presentado es generalizable. Así, si $H\Sigma$ es signatura oculta deconstructora, el conjunto de los contextos de género visible v para un género oculto h viene dado por:

$$\mathcal{C}_{H\Sigma}[z_h]_v = \{\sigma(z_h, d_\omega) \mid (\sigma\colon h\ \omega \to v) \in H\Sigma \text{ con } \omega \in VG^*,\ v \in VG \text{ y } d_\omega \in D_\omega\}$$

2.5.2 Caracterización de las Σ_{imp}-álgebras en el marco oculto

En esta sección vamos a establecer una caracterización de las signaturas obtenidas aplicando la operación $()_{imp}$ como un tipo particular de signaturas ocultas.

Sea $\Sigma = \langle G, \Omega \rangle$ una signatura y sea $\Sigma_{imp} = \left\langle G \cup \{imp_\Sigma\}, \{imp\_\sigma \colon imp_\Sigma\ \omega \to v\}_{(\sigma \colon \omega \to v) \in \Sigma} \right\rangle$ la signatura obtenida aplicando la operación $()_{imp}$ a Σ. Podemos interpretar Σ_{imp} como una signatura oculta. Para ello, consideramos como signatura de datos $V\Sigma = \langle G, \emptyset \rangle$ y fijamos un G-conjunto D sobre el universo algebraico $\mathcal{U}$. Por motivos técnicos, al igual que en el caso general, incluimos como operaciones constantes en la signatura $V\Sigma$ los elementos de D. Pues bien, podemos tomar $HG = \{imp_\Sigma\}$ y $H\Omega = \{imp\_\sigma \colon imp_\Sigma\ \omega \to v\}_{(\sigma \colon \omega \to v) \in \Sigma}$, lo que es fácil probar que nos da una descripción de Σ_{imp} como signatura oculta.

Remarcamos las siguientes propiedades de Σ_{imp}:

i) tiene un único género oculto, el género imp_Σ;

ii) no tiene operaciones visibles, y

iii) es una signatura deconstructora.

Las tres propiedades anteriores van a caracterizar a las signaturas que se obtienen a partir de la operación $()_{imp}$. A partir de una signatura oculta $H\Sigma$ sobre $(V\Sigma, D)$ verificando las tres propiedades anteriores, podemos construir, eliminando el único género oculto de $H\Sigma$, una signatura Σ, de modo que su signatura asociada Σ_{imp} es isomorfa (como signatura oculta) a $H\Sigma$. Esta forma de obtener Σ a partir de $H\Sigma$ corresponde a la definición de *interfaz para objetos*, como Cook apuntó en [29], página 169. El siguiente resultado recoge la caracterización que acabamos de comentar.

Proposición 2.5.9 *Sea $H\Sigma$ signatura oculta sobre $(V\Sigma, D)$ que posee un único género oculto, no tiene operaciones visibles y es deconstructora. Entonces existe una signatura Σ tal que Σ_{imp} y $H\Sigma$ son signaturas ocultas isomorfas.*

A partir del resultado anterior y de las propiedades vistas para las signaturas del tipo Σ_{imp}, se obtiene la siguiente caracterización.

Corolario 2.5.10 *Una signatura Δ se obtiene a partir de la operación $()_{imp}$ si y solo si existe una signatura oculta $H\Sigma$ con un único género oculto, sin operaciones visibles y deconstructora e isomorfa como signatura oculta a Δ.*

Ejemplo 2.5.11 Si consideramos la signatura `GRP` introducida en el ejemplo 2.2.1 y le aplicamos la operación $()_{imp}$, la signatura obtenida $\texttt{GRP}_{imp}$ es isomorfa como signatura oculta a la signatura $H\Sigma_2$ introducida en el ejemplo 2.5.2.

□

2.5.3 Existencia de objeto final en categorías de álgebras ocultas

Es bien conocido un resultado de existencia de álgebra final en categorías de álgebras ocultas ([46], [76], [27], entre otros), que se tiene como consecuencia de resultados del mismo tipo obtenidos por Burstall y Diaconescu en el campo de las álgebras comportamentales y que pueden encontrarse en [23].

Teorema 2.5.12 *Si $H\Sigma$ es una signatura oculta sobre $(V\Sigma, D)$ sin constructores a partir de datos, entonces la categoría $HAlg^D(H\Sigma)$ posee objeto final.*

En el resto de la sección nos restringiremos a trabajar con signaturas ocultas sin constructores a partir de datos.

En este punto podríamos preguntarnos cuál es el interés en estudiar un caso tan particular de signatura oculta sin constructores, como Σ_{imp}. Sin embargo, podemos ver que la misma objeción podríamos hacer respecto a las signaturas ocultas a las que se les puede aplicar el teorema anterior ya que sin constructores a partir de datos no somos capaces de construir ningún objeto. La idea subyacente es que los constructores a partir de datos son métodos especiales que deben estudiarse separadamente, algo que en los lenguajes de programación orientada a objeto está bien establecido. En nuestro caso particular, no aparecen operaciones de actualización ya que estamos interesados en el uso de la programación funcional pura. Los constructores a partir de objetos y datos se consideran aparte de las estructuras algebraicas concretas, ya que se ven como algoritmos o procesos externos que forman parte de un sistema completo. Así, nuestras signaturas solo recogen el comportamiento de aquellas estructuras a las que representan ya que los constructores están fuera. Es claro que, para completar la especificación de un sistema, deberíamos integrar signaturas estándar y signaturas Σ_{imp}, que solo tienen comportamiento, por medio de constructores, pero esto es una tarea distinta, y posterior.

2.5.3.1 El objeto final en $HAlg^D(H\Sigma)$

Una descripción explícita del objeto final en una categoría de álgebras ocultas puede encontrarse en [46] y [27], entre otros. Como Goguen y Malcolm expresan en [46], la descripción del álgebra oculta final viene expresada por "fórmulas mágicas" en las que se emplea la noción de contexto. Para introducir esa descripción del objeto final necesitamos dar previamente la siguiente definición.

Definición 2.5.13 *Sean $H\Sigma$ signatura oculta, h género oculto de $H\Sigma$, $c[z_h]_v$ un $H\Sigma$-contexto para el género h y t un $H\Sigma$-término. Si t es de género h, el $H\Sigma$-contexto $c[z_h]_v$ se dice* apropiado para el término t.

La idea intuitiva de contexto apropiado es que se puede sustituir en el contexto la variable por el término. Denotaremos por $c[t]_v$ al resultado de sustituir z_h por t en el contexto $c[z_h]_v$.

Si denotamos por $F^{H\Sigma}$ al álgebra final para la signatura oculta $H\Sigma$, una expresión para ella es la siguiente:

- Para cada género visible v se toma como soporte $F_v^{H\Sigma} = D_v$
- El soporte para cada género oculto $h \in HG$ viene dado por

$$F_h^{H\Sigma} = \prod_{v \in VG} D_v^{\mathcal{C}_{H\Sigma}[z_h]_v}$$

  Por tanto, cada $f \in F_h^{H\Sigma}$ es una tupla de funciones, indexada por el conjunto de géneros visibles. Luego, es de la forma $f = (f_v \colon \mathcal{C}_{H\Sigma}[z_h]_v \to D_v)_{v \in VG}$.

- Para cada $\sigma \colon \omega \to v$ en $V\Omega$, la interpretación $\sigma_{F^{H\Sigma}} \colon D_\omega \to D_v$ viene dada por $\sigma_{F^{H\Sigma}} := \sigma_D$.

- Para cada deconstructor $\sigma\colon h\ \omega \to v$ con $\omega \in VG^*$ y $v \in VG$, la operación $\sigma_{F^{H\Sigma}}\colon F_h^{H\Sigma} \times D_\omega \to D_v$ viene dada por

$$\sigma_{F^{H\Sigma}}\big((f_{v'})_{v'\in VG}, d_\omega\big) := f_v\big(\sigma(z_h, d_\omega)\big)$$

  para cada $(f_{v'}\colon \mathcal{C}_{H\Sigma}[z_h]_{v'} \to D_{v'})_{v'\in VG} \in F_h^{H\Sigma}$ y para cada $d_\omega \in D_\omega$.

  Para cada deconstructor σ, cada término de la forma $\sigma(z_h, d_\omega)$ es un contexto para el género h de género v y, por tanto, está en $\mathcal{C}_{H\Sigma}[z_h]_v$, por lo que la operación $\sigma_{F^{H\Sigma}}$ está bien definida.

- Para cada $\sigma\colon h\ \omega \to h'$ constructor a partir de objetos y datos u operación de actualización (es decir, $\omega \in VG^*$ y $h, h' \in HG$), hay que definir $\sigma_{F^{H\Sigma}}\colon F_h^{H\Sigma} \times D_\omega \to F_{h'}^{H\Sigma}$.

  Para cada $d_\omega \in D_\omega$, $\sigma(z_h, d_\omega)$ es un término de género h' por lo que es un término apropiado para cualquier contexto para el género h'. Por tanto, para cualquier $v \in VG$ y para cualquier contexto $c[z_{h'}] \in \mathcal{C}_{H\Sigma}[z_{h'}]_v$ de género v, podemos construir el $H\Sigma$-término $c[\sigma(z_h, d_\omega)]$ que, además, es un contexto de género v para el género h, por ser $c[z_{h'}]$ contexto.

  Definimos, para cada $(f_v\colon \mathcal{C}_{H\Sigma}[z_h]_v \to D_v)_{v\in VG} \in F_h^{H\Sigma}$ y para cada $d_\omega \in D_\omega$, $\sigma_{F^{H\Sigma}}\big((f_v)_{v\in VG}, d_\omega\big)$ como la tupla $(k_v\colon \mathcal{C}_{H\Sigma}[z_{h'}]_v \to D_v)_{v\in VG}$, siendo, para cada $v \in VG$ y para cada contexto $c[z_{h'}] \in \mathcal{C}_{H\Sigma}[z_{h'}]_v$, $k_v(c[z_{h'}]) := f_v(c[\sigma(z_h, d_\omega)])$.

En la descripción anterior puede observarse que la expresión del objeto final, en general, no es sencilla. Para ilustrar la definición del objeto, vamos a dar su expresión en el caso de las dos signaturas $H\Sigma_1$ y $H\Sigma_2$ introducidas en el ejemplo 2.5.2. Como ninguna de ellas posee constructores a partir de datos, el teorema 2.5.12 asegura la existencia del objeto final en las correspondientes categorías de álgebras ocultas. En el caso de la signatura $H\Sigma_2$ veremos que la expresión del álgebra final es más sencilla ya que, al ser deconstructora, los contextos tienen una expresión muy simple.

Ejemplo 2.5.14

1. En el caso de la signatura $H\Sigma_1$ los soportes para los géneros visibles son: $\mathbb{Z}$ para el género *int*, $\mathbb{N}$ para *nat* y $\mathcal{B} = \{true, false\}$ para *bool*.

   El soporte para el género oculto h es el siguiente conjunto:

$$F_h^{H\Sigma_1} = \mathbb{Z}^{\mathcal{C}_{H\Sigma_1}[z_h]_{int}} \times \mathbb{N}^{\mathcal{C}_{H\Sigma_1}[z_h]_{nat}} \times \mathcal{B}^{\mathcal{C}_{H\Sigma_1}[z_h]_{bool}}$$

Por tanto, cada $f \in F_h^{H\Sigma_1}$ podemos verla como una terna

$$f = \big(f_{int}\colon \mathcal{C}_{H\Sigma_1}[z_h]_{int} \to \mathbb{Z}, f_{nat}\colon \mathcal{C}_{H\Sigma_1}[z_h]_{nat} \to \mathbb{N}, f_{bool}\colon \mathcal{C}_{H\Sigma_1}[z_h]_{bool} \to \mathcal{B}\big)$$

La definición de las operaciones, en el caso de los deconstructores es clara:

· $abc_{F^{H\Sigma_1}}\colon F_h^{H\Sigma_1} \to \mathbb{Z}$ se define por $abc_{F^{H\Sigma_1}}\big((f_{int}, f_{nat}, f_{bool})\big) := f_{int}\big(abc(z_h)\big)$. La función está bien definida porque $abc(z_h)$ es un contexto de género *int* para el género h; análogamente,

· $ord_{F^{H\Sigma_1}}\colon F_h^{H\Sigma_1} \to \mathbb{N}$ viene dada por $ord_{F^{H\Sigma_1}}\big((f_{int}, f_{nat}, f_{bool})\big) := f_{nat}\big(ord(z_h)\big)$;

- $esx?_{F^{H\Sigma_1}}\colon F_h^{H\Sigma_1} \times \mathbb{Z} \to \mathcal{B}$ viene dada por $esx?_{F^{H\Sigma_1}}\big((f_{int}, f_{nat}, f_{bool}), z\big) := f_{bool}\big(esx?(z_h, z)\big)$, para todo $z \in \mathbb{Z}$. Está bien definida por ser $esx?(z_h, z)$ contexto de género *bool* para el género h; y, por último,
- $esy?_{F^{H\Sigma_1}}\colon F_h^{H\Sigma_1} \times \mathbb{N} \to \mathcal{B}$ se define por $esy?_{F^{H\Sigma_1}}\big((f_{int}, f_{nat}, f_{bool}), n\big) := f_{bool}\big(esy?(z_h, n)\big)$, para todo $n \in \mathbb{N}$.

Para definir las operaciones de actualización, como todas llegan al género oculto h, el resultado de la aplicación de cada operación va a ser una tupla de tres funciones, una por cada género visible:

- $move_{F^{H\Sigma_1}}\colon F_h^{H\Sigma_1} \times \mathbb{Z} \times \mathbb{N} \to F_h^{H\Sigma_1}$ viene dada, para cada $(f_{int}, f_{nat}, f_{bool}) \in F_h^{H\Sigma_1}$, para cada $z \in \mathbb{Z}$ y para cada $n \in \mathbb{N}$, por

  $$move_{F^{H\Sigma_1}}\big((f_{int}, f_{nat}, f_{bool}), z, n\big) := \big(f_{int}^{z,n}, f_{nat}^{z,n}, f_{bool}^{z,n}\big)$$

  siendo

  - $f_{int}^{z,n}\colon \mathcal{C}_{H\Sigma_1}[z_h]_{int} \to \mathbb{Z}$, función definida por $f_{int}^{z,n}(c[z_h]) := f_{int}(c[move(z_h, z, n)])$, para todo $c[z_h] \in \mathcal{C}_{H\Sigma_1}[z_h]_{int}$. Se tiene que $f_{int}^{z,n}$ está bien definida por ser $move(z_h, z, n)$ término apropiado para cualquier contexto de $\mathcal{C}_{H\Sigma_1}[z_h]_{int}$;
  - $f_{nat}^{z,n}\colon \mathcal{C}_{H\Sigma_1}[z_h]_{nat} \to \mathbb{N}$ viene dada por $f_{nat}^{z,n}(c[z_h]) := f_{nat}(c[move(z_h, z, n)])$, para todo $c[z_h] \in \mathcal{C}_{H\Sigma_1}[z_h]_{nat}$; y
  - $f_{bool}^{z,n}\colon \mathcal{C}_{H\Sigma_1}[z_h]_{bool} \to \mathcal{B}$ definida por $f_{bool}^{z,n}(c[z_h]) := f_{bool}(c[move(z_h, z, n)])$, para todo $c[z_h] \in \mathcal{C}_{H\Sigma_1}[z_h]_{bool}$;

- análogamente, la función $esc_{F^{H\Sigma_1}}\colon F_h^{H\Sigma_1} \times \mathbb{N} \to F_h^{H\Sigma_1}$ viene dada, para cada $(f_{int}, f_{nat}, f_{bool}) \in F_h^{H\Sigma_1}$ y para cada $n \in \mathbb{N}$ por $esc_{F^{H\Sigma_1}}\big((f_{int}, f_{nat}, f_{bool}), n\big) := (f_{int}^n, f_{nat}^n, f_{bool}^n)$ siendo $f_i^n(c[z_h]) := f_i(c[esc(z_h, n)])$, para todo $c[z_h] \in \mathcal{C}_{H\Sigma_1}[z_h]_i$ y para $i \in \{int, nat, bool\}$; y, finalmente,
- $sim_{F^{H\Sigma_1}}\colon F_h^{H\Sigma_1} \to F_h^{H\Sigma_1}$ se define, para cada $(f_{int}, f_{nat}, f_{bool}) \in F_h^{H\Sigma_1}$, como $sim_{F^{H\Sigma_1}}\big((f_{int}, f_{nat}, f_{bool})\big) := (f_{int}^*, f_{nat}^*, f_{bool}^*)$ siendo $f_i^*(c[z_h]) := f_i(c[sim(z_h)])$, para todo $c[z_h] \in \mathcal{C}_{H\Sigma_1}[z_h]_i$ y todo $i \in \{int, nat, bool\}$.

2. En el caso de la signatura $H\Sigma_2$, al ser deconstructora la descripción del objeto final es más sencilla. El soporte para el género visible v es $\mathbb{Z}$ y, para el género oculto h el soporte viene dado por $F_h^{H\Sigma_2} = \mathbb{Z}^{\mathcal{C}_{H\Sigma_2}[z_h]}$.

   Las operaciones se definen como sigue:

   - $prd\_grp_{F^{H\Sigma_2}}\colon F_h^{H\Sigma_2} \times \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ definida, para cada $f \in F_h^{H\Sigma_2}$ y para cada $z_1, z_2 \in \mathbb{Z}$, por $prd\_grp_{F^{H\Sigma_2}}(f, z_1, z_2) := f(prd\_grp(z_h, z_1, z_2))$. Como $prd\_grp(z_h, z_1, z_2)$ es un contexto para el género h de género v, la función está bien definida;
   - $inv\_grp_{F^{H\Sigma_2}}\colon F_h^{H\Sigma_2} \times \mathbb{Z} \to \mathbb{Z}$ definida por $inv\_grp_{F^{H\Sigma_2}}(f, z_1) := f(inv\_grp(z_h, z_1))$, para cada $f \in F_h^{H\Sigma_2}$ y para cada $z_1 \in \mathbb{Z}$; y,
   - $unt\_grp_{F^{H\Sigma_2}}\colon F_h^{H\Sigma_2} \to \mathbb{Z}$ definida por $unt\_grp_{F^{H\Sigma_2}}(f) := f(unt\_grp(z_h))$, para cada $f \in F_h^{H\Sigma_2}$.

□

2.5.3.2 El morfismo final en $HAlg^D(H\Sigma)$

Para cualquier $H\Sigma$-álgebra oculta A, la expresión del morfismo final $j\colon A \to F^{H\Sigma}$ es la natural:

- $j_v = id_{D_v}$, para cada $v \in VG$, y
- para cada género oculto h, $j_h\colon A_h \to F_h^{H\Sigma}$ definida, para cada $a \in A_h$, por la familia de aplicaciones $j_h(a) := (f_v^a\colon \mathcal{C}_\Sigma[z_h]_v \to D_v)_{v\in VG}$ siendo, para cada $c[z_h] \in \mathcal{C}_\Sigma[z_h]_v$, $f_v^a(c[z_h]) := A_c[a]$, donde $A_c(a)$ denota el resultado de interpretar $c[z_h]$ en A sustituyendo la variable z_h por a.

Vamos a detallar este morfismo en los ejemplos anteriores, para las signaturas ocultas $H\Sigma_1$ y $H\Sigma_2$.

Ejemplo 2.5.15

1. Sea $A = \langle \mathbb{Z}\times\mathbb{N}, \{abc_A, ord_A, esx?_A, esy?_A, move_A, esc_A, inv_A\}\rangle$ una $H\Sigma_1$-álgebra oculta sobre $(V\Sigma_1, \{\mathbb{Z}, \mathbb{N}, \mathcal{B}\})$.

   El morfismo final $j\colon A \to F^{H\Sigma_1}$ viene dado, por la identidad sobre los soportes de los géneros visibles y, para el género oculto h, $j_h\colon \mathbb{Z}\times\mathbb{N} \to F_h^{H\Sigma_1}$ se define por $j_h\big((x,y)\big) := \big(f_{int}^{(x,y)}, f_{nat}^{(x,y)}, f_{bool}^{(x,y)}\big)$ estando definidas las funciones anteriores del siguiente modo:

   - $f_{int}^{(x,y)}\colon \mathcal{C}_{H\Sigma_1}[z_h]_{int} \to \mathbb{Z}$ dada por:
     - $f_{int}^{(x,y)}(abc(z_h)) := abc_A((x,y)) = x$
     - $f_{int}^{(x,y)}(abc(met_n(\dots(met_1(z_h, d_{\omega_1}),\dots), d_{\omega_n}))) :=$
       $= abc_A(met_{n_A}(\dots(met_{1_A}((x,y), d_{\omega_1}),\dots), d_{\omega_n}))$
   - $f_{nat}^{(x,y)}\colon \mathcal{C}_{H\Sigma_1}[z_h]_{nat} \to \mathbb{N}$ dada por:
     - $f_{nat}^{(x,y)}(ord(z_h)) := ord_A((x,y)) = y$
     - $f_{nat}^{(x,y)}(ord(met_n(\dots(met_1(z_h, d_{\omega_1}),\dots), d_{\omega_n}))) :=$
       $= ord_A(met_{n_A}(\dots(met_{1_A}((x,y), d_{\omega_1}),\dots), d_{\omega_n}))$
   - $f_{bool}^{(x,y)}\colon \mathcal{C}_{H\Sigma_1}[z_h]_{bool} \to \mathcal{B}$ dada por:
     - $f_{bool}^{(x,y)}(esx?(z_h, d_z)) := esx?_A((x,y), d_z) = \begin{cases} true & \text{si } x = d_z \\ false & \text{en otro caso} \end{cases}$
     - $f_{bool}^{(x,y)}(esy?(z_h, d_n)) := esy?_A((x,y), d_n) = \begin{cases} true & \text{si } y = d_n \\ false & \text{en otro caso} \end{cases}$
     - $f_{bool}^{(x,y)}(esx?(met_n(\dots(met_1(z_h, d_{\omega_1}),\dots), d_{\omega_n}), d_z)) :=$
       $= esx?_A(met_{n_A}(\dots(met_{1_A}((x,y), d_{\omega_1}),\dots), d_{\omega_n}), d_z)$

$$\cdot\ f_{bool}^{(x,y)}(esy?(met_n(\dots(met_1(z_h,d_{\omega_1}),\dots),d_{\omega_n}),d_n)) := \\ = esy?_A(met_{n_A}(\dots(met_{1_A}((x,y),d_{\omega_1}),\dots),d_{\omega_n}),d_n)$$

con $d_z \in \mathbb{Z}$, $d_n \in \mathbb{N}$, y para cada $i \in \{1,\dots n\}$, $met_i \in \{move, esc, sim\}$ con

$$d_{\omega_i} \in \begin{cases} \mathbb{Z}\times\mathbb{N} & \text{si } met_i = move \\ \mathbb{N} & \text{si } met_i = esc \\ \text{nada} & \text{en otro caso } (met_i = sim) \end{cases}$$

2. Si $B = \langle B_h, \{prd\_grp_B, inv\_grp_B, unt\_grp_B\}\rangle$ es una $H\Sigma_2$-álgebra oculta sobre $\mathbb{Z}$, el morfismo final $k\colon B \to F^{H\Sigma_2}$ viene dado, para el género oculto h, por la aplicación $k_h\colon B_h \to F_h^{H\Sigma_2}$ siendo para cada $b \in B_h$, $k_h(b)\colon \mathcal{C}_{H\Sigma_2}[z_h] \to \mathbb{Z}$ la aplicación definida por:

   - $k_h(b)(prd\_grp(z_h,z_1,z_2)) := prd\_grp_B(b,z_1,z_2)$
   - $k_h(b)(inv\_grp(z_h,z_1)) := inv\_grp_B(b,z_1)$
   - $k_h(b)(unt\_grp(z_h)) := unt\_grp_B(b)$,

   para todo $z_1, z_2 \in \mathbb{Z}$.

□

2.5.4 Las categorías $HAlg^D(\Sigma_{imp})$ y $Alg^{D,\{\}}(\Sigma_{imp})$

Sea $\Sigma = \langle G, \Omega\rangle$ una signatura y fijamos un G-conjunto D sobre el universo algebraico $\mathcal{U}$. Podemos considerar dos categorías de álgebras relacionadas con la signatura anterior. Por una parte, la categoría que hemos denotado en la sección anterior por $Alg^{D,\{\}}(\Sigma_{imp})$ compuesta por las Σ_{imp}-álgebras con soporte D_g para cada g género de Σ y con morfismos los Σ_{imp}-morfismos que son la identidad sobre D. Por otra parte, si interpretamos Σ_{imp} como signatura oculta sobre $(\langle G, \emptyset\rangle, D)$, podemos considerar la categoría de álgebras ocultas $HAlg^D(\Sigma_{imp})$.

Los objetos y los morfismos de ambas categorías son los mismos, aunque interpretados en distintos marcos. Así, las categorías $HAlg^D(\Sigma_{imp})$ y $Alg^{D,\{\}}(\Sigma_{imp})$ pueden identificarse.

Como Σ_{imp} es deconstructora, en particular no posee constructores a partir de datos, por lo que el teorema 2.5.12 asegura la existencia de objeto final en $HAlg^D(\Sigma_{imp})$. Este resultado no nos debe sorprender, puesto que el teorema 2.4.6 asegura la existencia de objeto final en $Alg^{D,\{\}}(\Sigma_{imp})$. Por tanto, debe existir un isomorfismo entre el objeto $\mathbb{1}^D$, descrito en la sección anterior, y el objeto $F^{\Sigma_{imp}}$. Esto hará que podamos trabajar indistintamente con cualquiera de ellos, aunque es evidente que la expresión del objeto $\mathbb{1}^D$ es más sencilla. En cualquier caso, nos parece interesante dar explícitamente ese isomorfismo. Recordemos las descripciones de los objetos:

- Aplicamos la descripción general vista para el objeto final en una categoría de álgebras ocultas. En el caso de la categoría $HAlg^D(\Sigma_{imp})$ la expresión es más sencilla, ya que $\Sigma_{imp} = \langle G \sqcup \{imp_\Sigma\}, \{imp\_\sigma\colon imp_\Sigma\ \omega \to v\}_{(\sigma\colon \omega\to v)\in\Sigma}\rangle$ solo posee un género oculto y

es deconstructora. Denotando por $\mathbb{F}$ al objeto final en $HAlg^D(\Sigma_{imp})$ (anteriormente lo denotamos $F^{\Sigma_{imp}}$, notación que abreviamos ahora por simplificarla), su expresión es la siguiente:

- Para cada $v \in G$: $\mathbb{F}_v = D_v$
- El soporte para el género oculto imp_Σ viene dado por

$$\mathbb{F}_{imp_\Sigma} = \prod_{v \in G} D_v^{\mathcal{C}_{\Sigma_{imp}}[z_{imp_\Sigma}]_v}$$

- Para cada deconstructor $imp\_\sigma\colon imp_\Sigma\ \omega \to v$ con $\omega \in G^*$ y $v \in G$, la operación $imp\_\sigma_{\mathbf{F}}\colon \mathbb{F}_{imp_\Sigma} \times D_\omega \to D_v$ viene dada por

$$imp\_\sigma_{\mathbf{F}}\big((f_{v'})_{v' \in G}, d_\omega\big) := f_v\big(imp\_\sigma(z_{imp_\Sigma}, d_\omega)\big)$$

para todo $(f_{v'}\colon \mathcal{C}_{\Sigma_{imp}}[z_{imp_\Sigma}]_{v'} \to D_{v'})_{v' \in G} \in \mathbb{F}_{imp_\Sigma}$ y para todo $d_\omega \in D_\omega$.

- La primera descripción que dimos del objeto final $\mathbb{1}^D$ en $Alg^{D,\{\}}(\Sigma_{imp})$ fue verlo como la aplicación identidad $\mathbb{1}^D\colon Alg^D(\Sigma) \to Alg^D(\Sigma)$. Posteriormente vimos otra descripción del objeto como tuplas de funciones, expresión que viene dada por:

$$\Big\langle \prod_{(\sigma\colon \omega' \to v') \in \Sigma} D_{v'}^{D_{\omega'}}, D, \{imp\_\sigma_{\mathbb{1}^D}\colon \prod_{(\sigma\colon \omega' \to v') \in \Sigma} D_{v'}^{D_{\omega'}} \times D_\omega \to D_v\}_{(\sigma\colon \omega \to v) \in \Sigma} \Big\rangle$$

siendo $imp\_\sigma_{\mathbb{1}^D}\big((f_\delta)_{(\delta\colon \omega' \to v') \in \Sigma}, d_\omega\big) := f_\sigma(d_\omega)$, para todo $(f_\delta)_{(\delta\colon \omega' \to v') \in \Sigma} \in \prod_{(\sigma\colon \omega' \to v') \in \Sigma} D_{v'}^{D_{\omega'}}$ y para todo $d_\omega \in D_\omega$.

La siguiente proposición relaciona los soportes de ambas Σ_{imp}-álgebras.

Proposición 2.5.16 *Sea $H\Sigma$ signatura oculta deconstructora sobre $(V\Sigma, D)$, con un género oculto h y sin operaciones visibles. Existe una biyección entre los conjuntos*

$$\mathbb{A} = \prod_{v \in VG} D_v^{\mathcal{C}_{H\Sigma}[z_h]_v} \quad \text{y} \quad \mathbb{A}' = \prod_{(\sigma\colon h\ \omega \to v) \in H\Omega} D_v^{D_\omega}$$

Demostración:

La idea de la demostración es que cada aplicación $f_v\colon \mathcal{C}_{H\Sigma}[z_h]_v \to D_v$ puede descomponerse como una tupla de aplicaciones, cada una de ellas correspondiente a una operación de $H\Omega$ de género v, sin más que hacer una partición del conjunto $\mathcal{C}_{H\Sigma}[z_h]_v$ de forma que se agrupen todos los contextos procedentes del mismo símbolo de operación.

Por ser $H\Sigma$ deconstructora, para cada género visible $v \in VG$, el conjunto de contextos para el género h es el siguiente:

$$\mathcal{C}_{H\Sigma}[z_h]_v = \big\{\sigma(z_h, d_\omega)\ \mid\ (\sigma\colon h\ \omega \to v) \in H\Sigma \text{ con } \omega \in VG^*,\ v \in VG \text{ y } d_\omega \in D_\omega\big\}$$

El conjunto anterior podemos verlo como una unión disjunta, agrupando aquellos contextos que provienen de la misma operación $(\sigma\colon h\ \omega \to v)$ de $H\Omega$. Así, si denotamos por $\mathcal{C}_{H\Sigma}[z_h]_v^\sigma$ al conjunto

$$\mathcal{C}_{H\Sigma}[z_h]_v^\sigma = \big\{\sigma(z_h, d_\omega)\ \mid\ d_\omega \in D_\omega\big\}$$

es claro que $\mathcal{C}_{H\Sigma}[z_h]^\sigma_v$ es biyectivo con el conjunto D_ω.

Además, recorriendo todas las operaciones de $H\Omega$ se tiene que

$$\mathcal{C}_{H\Sigma}[z_h]_v \cong \bigsqcup_{(\sigma:\, h\ \omega\to v)\in H\Sigma} \mathcal{C}_{H\Sigma}[z_h]^\sigma_v \cong \bigsqcup_{(\sigma:\, h\ \omega\to v)\in H\Sigma} D_\omega$$

Aplicando ahora la propiedad universal del coproducto en conjuntos, se obtiene la biyección entre $\mathbb{A}$ y $\mathbb{A}'$.

■

Considerando Σ_{imp} como signatura oculta, el conjunto $\mathbb{A}$ de la proposición anterior es el soporte para el género oculto en el objeto final $F^{\Sigma_{imp}}$ (o $\mathbb{F}$ como lo hemos denotado anteriormente), mientras que el conjunto $\mathbb{A}'$ es el soporte para el género distinguido imp_Σ en el objeto $\mathbb{1}^D$. Es sencillo probar que la biyección anterior define un isomorfismo entre ambos objetos (basta comprobar que, junto con las identidades sobre D, define un morfismo).

Vamos a estudiar el isomorfismo anterior para el caso de la signatura $\mathtt{GRP}_{imp}$.

Ejemplo 2.5.17 Para cada conjunto D, el objeto final de la categoría $HAlg^D(\mathtt{GRP}_{imp})$, al que denominamos $\mathbb{F}$, admite una descripción que corresponde a tomar el espacio de funciones $D^{\mathcal{C}[z_{imp_{\mathtt{GRP}}}]}$ como soporte para el género $imp_{\mathtt{GRP}}$ y definir las operaciones como sigue:

- $imp\_prd_{\mathbf{F}}(f, d_1, d_2) := f(prd\_grp(z_h, d_1, d_2))$,
- $imp\_inv_{\mathbf{F}}(f, d_1) := f(inv\_grp(z_h, d_1))$,
- $imp\_unt_{\mathbf{F}}(f) := f(unt\_grp(z_h))$;

para todo $f \in D^{\mathcal{C}[z_{imp_{\mathtt{GRP}}}]}$ y para todo $d_1, d_2 \in D$.

Observar que, en este caso, el conjunto de contextos $\mathcal{C}[z_{imp_{\mathtt{GRP}}}]$ admite una descomposición

$$\mathcal{C}[z_{imp_{\mathtt{GRP}}}] = \mathcal{C}[z_{imp_{\mathtt{GRP}}}]^{imp\_prd} \sqcup \mathcal{C}[z_{imp_{\mathtt{GRP}}}]^{imp\_inv} \sqcup \mathcal{C}[z_{imp_{\mathtt{GRP}}}]^{imp\_unt}$$

siendo

- $\mathcal{C}[z_{imp_{\mathtt{GRP}}}]^{imp\_prd} = \{imp\_prd(z_{imp_{\mathtt{GRP}}}, d_1, d_2) \mid d_1, d_2 \in D\}$,
- $\mathcal{C}[z_{imp_{\mathtt{GRP}}}]^{imp\_inv} = \{imp\_inv(z_{imp_{\mathtt{GRP}}}, d_1) \mid d_1 \in D\}$ y
- $\mathcal{C}[z_{imp_{\mathtt{GRP}}}]^{imp\_prd} = \{imp\_unt(z_{imp_{\mathtt{GRP}}})\}$.

Así, es claro que el conjunto $D^{\mathcal{C}[z_{imp_{\mathtt{GRP}}}]}$ es biyectivo con $D^{\mathcal{C}[z_{imp_{\mathtt{GRP}}}]^{imp\_prd}} \times D^{\mathcal{C}[z_{imp_{\mathtt{GRP}}}]^{imp\_inv}} \times D^{\mathcal{C}[z_{imp_{\mathtt{GRP}}}]^{imp\_unt}}$.

Además, el conjunto $\mathcal{C}[z_{imp_{\mathtt{GRP}}}]^{imp\_prd}$ es biyectivo con $D \times D$, $\mathcal{C}[z_{imp_{\mathtt{GRP}}}]^{imp\_inv}$ lo es con D y $\mathcal{C}[z_{imp_{\mathtt{GRP}}}]^{imp\_unt}$ es un conjunto unipuntual.

Para cada conjunto D, la expresión del objeto final $\mathbb{1}^D_{\mathtt{GRP}_{imp}}$ de la categoría $Alg^{D,\{\}}(\Sigma_{imp})$ en el ejemplo 2.4.3, corresponde con la siguiente $\mathtt{GRP}_{imp}$-álgebra:

$$\left\langle D^{D\times D} \times D^D \times D, D, \{imp\_prd_{\mathbb{1}^D_{\mathtt{GRP}_{imp}}}, imp\_inv_{\mathbb{1}^D_{\mathtt{GRP}_{imp}}}, imp\_unt_{\mathbb{1}^D_{\mathtt{GRP}_{imp}}}\}\right\rangle$$

siendo

- $imp\_prd_{\mathbb{1}^D_{\mathtt{GRP}_{imp}}}((f_{prd}, f_{inv}, d_0), d_1, d_2) := f_{prd}(d_1, d_2)$,
- $imp\_inv_{\mathbb{1}^D_{\mathtt{GRP}_{imp}}}((f_{prd}, f_{inv}, d_0), d_1) := f_{inv}(d_1)$,
- $imp\_unt_{\mathbb{1}^D_{\mathtt{GRP}_{imp}}}((f_{prd}, f_{inv}, d_0)) := d_0$;

para todo $(f_{prd}, f_{inv}, d_0) \in D^{D\times D} \times D^D \times D$ y para todo $d_1, d_2 \in D$.

Por tanto, el conjunto $D^{\mathcal{C}[z_{imp_{\mathtt{GRP}}}]}$ es biyectivo con $D^{D\times D} \times D^D \times D$. Por último, es claro que las operaciones de $\mathbb{F}$ se transforman por esta biyección en las de $\mathbb{1}^D_{\mathtt{GRP}_{imp}}$.

□

2.6 Visión coalgebraica

Otro de los conceptos matemáticos utilizados en las dos últimas décadas para el tratamiento formal de estructuras de datos es el de coálgebra, dual de la noción de álgebra. La diferencia esencial entre álgebras y coálgebras está en que las primeras poseen constructores, es decir, operaciones que llegan al género distinguido, mientras que las segundas poseen observadores, operaciones que salen del género distinguido y que permiten observar determinado comportamiento. De ahí la adecuación de las coálgebras para su aplicación en el paradigma orientado a objeto. La diferencia entre construir y observar comportamiento es la esencia de la distinción entre tipos abstractos de datos y la abstracción procedural descrita por Cook en [29].

En [62] y [76] se describen relaciones entre los marcos algebraico y coalgebraico, viendo que, en muchas ocasiones, son complementarios.

El primero en aplicar la aproximación coalgebraica a la formalización de estructuras de datos fue Reichel en [91] en 1995. Su construcción se ha utilizado en otros trabajos ([76] y [27], entre otros) mostrando que añadiendo constructores a partir de datos a una coálgebra se obtiene un álgebra oculta. Esto hace que ambas aproximaciones, la de especificaciones ocultas y la coalgebraica, estén muy cercanas. Esta última, al igual que la especificación oculta, tiene su aplicación en el paradigma orientado a objeto y permite especificar estructuras de datos infinitas.

En la práctica, una de las características de la aproximación coalgebraica es que se trabaja con signaturas en cuyas operaciones aparece el género distinguido como argumento una única vez. En general, la codificación de objetos infinitos no se aborda desde el ámbito de la especificación algebraica tradicional, ya que el álgebra de términos no describe adecuadamente la semántica de los objetos infinitos. Sin embargo, muchas de las signaturas que provienen de estructuras de datos que codifican objetos infinitos verifican la propiedad de que el género distinguido aparece una única vez como argumento en todas las operaciones, por ello, como Rutten comenta en [106], la aproximación coalgebraica se emplea como alternativa al marco algebraico para formalizar estructuras de datos que codifican objetos infinitos y cuyas signaturas asociadas verifican la propiedad anterior. (En [81] se aborda la especificación de objetos infinitos pero en otro marco distinto.)

Introducimos a continuación conceptos básicos sobre coálgebras para seguidamente mostrar cómo la aproximación coalgebraica cubre el caso de las Σ_{imp}-álgebras. Recogeremos la relación con las especificaciones ocultas y, finalmente recopilaremos la relación existente entre todos los

marcos presentados (constucción $()_{Set}$, especificaciones ocultas y aproximación coalgebraica) en el caso de trabajar con signaturas Σ_{imp}.

2.6.1 Concepto de coálgebra

En la literatura pueden encontrarse trabajos generales sobre la teoría de coálgebras (por ejemplo, [2] y [8]), desarrollados dentro del Álgebra Universal. En [106], Rutten desarrolla una teoría de coálgebras paralela a la teoría de álgebras en Álgebra Universal pero enfocada a su aplicación en el campo de la especificación de tipos de datos. Las nociones introducidas por Rutten en [106] son las mismas que las empleadas en otros trabajos que siguen sus mismos objetivos (ver [91], [60], [61]) y son, frecuentemente, nociones más particulares que las que pueden encontrarse en los trabajos sobre coálgebras en general. En esta sección pretendemos mostrar la adecuación del marco coalgebraico para la especificación y formalización de determinadas estructuras de datos, por lo que adoptaremos las nociones utilizadas en [106], [91] y [60], entre otros, por ser de aplicación más directa a nuestro trabajo.

Como es habitual, *Set* denota a la categoría de los conjuntos y las aplicaciones totales.

Definición 2.6.1 *Sea $F\colon Set \to Set$ un endofuntor de Set. Una* F-coálgebra *es un par (A, α_A) donde A es un conjunto y $\alpha_A\colon A \to F(A)$ es una aplicación.*

Si no hay lugar a confusión, hablaremos simplemente de coálgebra sin hacer referencia al endofuntor de *Set* y escribiremos $\alpha_A\colon A \to F(A)$ para referirnos a la F-coálgebra (A, α_A).

El siguiente paso es introducir la noción de morfismo de coálgebras.

Definición 2.6.2 *Sea $F\colon Set \to Set$ endofuntor de Set. Un* morfismo entre dos F-coálgebras (A, α_A) y (B, α_B) *es una aplicación $f\colon A \to B$ tal que $F(f) \circ \alpha_A = \alpha_B \circ f$.*

$$\begin{array}{ccc} A & \xrightarrow{\alpha_A} & F(A) \\ {\scriptstyle f}\downarrow & \circlearrowleft & \downarrow{\scriptstyle F(f)} \\ B & \xrightarrow[\alpha_B]{} & F(B) \end{array}$$

Para cada F endofuntor de *Set*, las F-coálgebras junto con los morfismos entre ellas definen una categoría que denotaremos por $CoAlg(F)$.

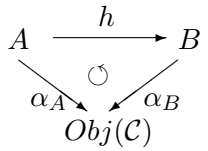
Existe una generalización natural de la noción de coálgebra, la extensión de la definición a cualquier endofuntor de una categoría $\mathcal{C}$ (esta definición es la adoptada en trabajos como [2] y [8], dedicados al estudio del marco coalgebraico en sí).

2.6.2 Interpretación de las familias de álgebras como coálgebras

Para cada categoría pequeña $\mathcal{C}$ vamos a considerar un funtor, que denotaremos por $F_{\mathcal{C}}$, de forma que la categoría de coálgebras que define está muy relacionada con la categoría $\mathcal{C}_{Set}$. Más concretamente, veremos que la categoría de coálgebras definida por $F_{\mathcal{C}}$ es una subcategoría muy particular de $\mathcal{C}_{Set}$.

Sea $\mathcal{C}$ una categoría pequeña. Consideramos el endofuntor de Set constante sobre el conjunto $Obj(\mathcal{C})$ (véase definición en preliminares, página 20) y lo denotamos por $F_{\mathcal{C}}$. Así, una $F_{\mathcal{C}}$-coálgebra es una aplicación $\alpha_A \colon A \to Obj(\mathcal{C})$. Es evidente que $F_{\mathcal{C}}$-coálgebras y objetos de $\mathcal{C}_{Set}$ son una misma cosa. Por tanto, los objetos de las categorías $CoAlg(F_{\mathcal{C}})$ y $\mathcal{C}_{Set}$ pueden identificarse.

Por su parte, un morfismo entre dos $F_{\mathcal{C}}$-coálgebras $\alpha_A \colon A \to Obj(\mathcal{C})$ y $\alpha_B \colon B \to Obj(\mathcal{C})$ es una aplicación $h \colon A \to B$ que hace el triángulo correspondiente conmutativo, es decir, verifica que $\alpha_A = \alpha_B \circ h$.



Notar que estos morfismos corresponden con los morfismos de la subcategoría de $\mathcal{C}_{Set}$ que hemos denotado por $\mathcal{C}_{Set}^{\{\}}$. Por tanto, las categorías $CoAlg(F_{\mathcal{C}})$ y $\mathcal{C}_{Set}^{\{\}}$ se identifican de manera natural, lo que permite interpretar cada familia indexada de objetos de $\mathcal{C}$ como una $F_{\mathcal{C}}$-coálgebra. Además, esto nos permite asegurar que $CoAlg(F_{\mathcal{C}})$ tiene objeto final, el objeto $\mathbb{1} \colon Obj(\mathcal{C}) \to Obj(\mathcal{C})$.

Volviendo a las Σ-álgebras, si $\Sigma = \langle V, \Omega \rangle$ es una signatura y D un V-conjunto, consideramos la categoría de Σ-álgebras con soporte D, $Alg^D(\Sigma)$. Considerando el correspondiente funtor constante $F_{Alg^D(\Sigma)}$, también se identifican las categorías $CoAlg(F_{Alg^D(\Sigma)})$ y $Alg^{D,\{\}}(\Sigma)_{Set}$, siendo ésta última la categoría de las familias indexadas de Σ-álgebras con soporte D. Por tanto, cada familia indexada de Σ-álgebras con soporte D, puede interpretarse como una $F_{Alg^D(\Sigma)}$-coálgebra.

Si a lo anterior añadimos el isomorfismo existente entre $Alg^{D,\{\}}(\Sigma)_{Set}$ y $Alg^{D,\{\}}(\Sigma_{imp})$, podemos concluir que las categorías $CoAlg(F_{Alg^D(\Sigma)})$ y $Alg^{D,\{\}}(\Sigma_{imp})$ son isomorfas, y, así, cada Σ_{imp}-álgebra con soporte D puede interpretarse como una $F_{Alg^D(\Sigma)}$-coálgebra.

Llamaremos coálgebra final asociada a una signatura Σ y a un V-conjunto D al objeto final de la categoría $CoAlg(F_{Alg^D(\Sigma)})$. Una descripción de esta coálgebra final, obtenida directamente de la propia definición de coálgebra, es $\mathbb{1}^D \colon Alg^D(\Sigma) \to Alg^D(\Sigma)$, que coincide exactamente con la descripción dada anteriormente para el objeto final de la categoría $Alg^{D,\{\}}(\Sigma)_{Set}$.

La biyección existente entre los conjuntos $Alg^D(\Sigma)$ y $\prod_{(\sigma \colon \omega \to v) \in \Sigma} D_v^{D_\omega}$, hace que el funtor constante $F_\Sigma \colon Set \to Set$ sobre el conjunto $\prod_{(\sigma \colon \omega \to v) \in \Sigma} D_v^{D_\omega}$ sea equivalente al funtor $F_{Alg^D(\Sigma)}$, lo que a su vez implica que sus correspondientes categorías de coálgebras, $CoAlg(F_\Sigma)$ y $CoAlg(F_{Alg^D(\Sigma)})$, sean isomorfas.

La aplicación identidad sobre el conjunto $\prod_{(\sigma \colon \omega \to v) \in \Sigma} D_v^{D_\omega}$ puede verse como otra descripción de la coálgebra final asociada a la signatura $\Sigma = \langle V, \Omega \rangle$ y al V-conjunto D. Por tanto, la aplicación $id_{\prod_{(\sigma \colon \omega \to v) \in \Sigma} D_v^{D_\omega}}$ cubre a la familia de todas las Σ-álgebras con soporte D, resultado que ya se había obtenido por otra vía.

2.6.3 Categoría de coálgebras asociada a una signatura (oculta)

La relación entre especificaciones ocultas y coálgebras ha sido estudiada en diversos trabajos ([76] y [27] entre otros). Informalmente, las álgebras ocultas son más generales que las

coálgebras, en el sentido de que toda coálgebra da lugar a un álgebra oculta para una cierta signatura oculta. El recíproco, es decir, interpretar un álgebra oculta como una coálgebra, no es posible en general, sin embargo, sí lo es en el caso de trabajar con signaturas ocultas que poseen un único género oculto y sin constructores a partir de datos. Las signaturas Σ_{imp}, objeto de nuestro estudio, vistas como signaturas ocultas, poseen las propiedades anteriores, por lo que son susceptibles de ser tratadas como coálgebras.

Vamos a asociar una categoría de coálgebras a una signatura oculta que posea las propiedades requeridas. Para ello, a partir de la signatura vamos a definir un endofuntor de Set, cuya categoría de coálgebras será la que diremos que está asociada a la signatura de partida.

Partimos de una signatura $V\Sigma = \langle VG, \emptyset \rangle$ y de un álgebra de datos $D = (D_v)_{v \in VG}$. Sea $H\Sigma = \langle VG \sqcup \{h\}, H\Omega \rangle$ una signatura oculta sobre $(V\Sigma, D)$ que no posee constructores a partir de datos.

Por tanto, estamos considerando una signatura oculta que verifica las tres siguientes propiedades:

i) no posee constructores a partir de datos;

ii) posee un único género oculto, y

iii) no posee operaciones visibles.

La condición *iii*) es superflua puesto que, como ya hemos comentado en la sección anterior, al incluir los elementos del álgebra de datos como constantes en la signatura, las constantes proporcionan un sistema completo de representantes de términos variables y, además, sin identificaciones entre ellos. Así, se puede trabajar suponiendo que no hay operaciones visibles, condición que imponemos simplemente por comodidad, nos evita considerar términos visibles no constantes.

Podemos interpretar una signatura oculta verificando las tres propiedades anteriores como una signatura clásica, viendo el género oculto como género distinguido y tomando el álgebra de datos D como VG-conjunto fijo (los deconstructores pasarían a ser observadores y los operadores de actualización serán constructores). Observar que las signaturas obtenidas por la operación $()_{imp}$ son de este tipo.

En toda esta sección, cada vez que nos refiramos a una signatura oculta supondremos que verifica las tres propiedades anteriores. Como venimos haciendo, utilizaremos el símbolo h para denotar al único género oculto y nos referiremos a una signatura oculta $H\Sigma = \langle VG \sqcup \{h\}, H\Omega \rangle$ sobre $(V\Sigma, D)$, simplemente por signatura oculta sobre D. De igual modo, el símbolo v lo emplearemos para denotar géneros visibles, así un deconstructor lo escribiremos $\sigma\colon h\ \omega \to v$ con $\omega \in VG^*$, y un operador de actualización $\sigma\colon h\ \omega \to h$.

A partir de una signatura oculta $H\Sigma$ sobre D, vamos a definir un endofuntor de la categoría de conjuntos.

Definición 2.6.3 *Llamaremos* funtor asociado a la signatura oculta $H\Sigma = \langle VG \sqcup \{h\}, H\Omega \rangle$ sobre D, *lo denotamos por* $F_{H\Sigma}\colon Set \to Set$, *al endofuntor de Set definido, para cada conjunto* X, *por*

$$F_{H\Sigma}(X) := \prod_{(\sigma\colon h\ \omega \to v) \in H\Sigma} D_v^{D_\omega} \times \prod_{(\sigma\colon h\ \omega \to h) \in H\Sigma} X^{D_\omega}$$

y, para cada aplicación $f\colon X \to Y$, $F_{H\Sigma}(f)\colon F_{H\Sigma}(X) \to F_{H\Sigma}(Y)$ *es la aplicación que a cada* $g \in F_{H\Sigma}(X)$, *siendo* $g = \big((g_\sigma\colon D_\omega \to D_v)_{(\sigma\colon h\ \omega\to v)\in H\Sigma}, (g_\sigma\colon D_\omega \to X)_{(\sigma\colon h\ \omega\to h)\in H\Sigma}\big)$, *lo manda sobre* $F_{H\Sigma}(f)(g) := \big((g_\sigma)_{(\sigma\colon h\ \omega\to v)\in H\Sigma}, (f\circ g_\sigma)_{(\sigma\colon h\ \omega\to h)\in H\Sigma}\big)$, *es decir, definida como la identidad sobre las componentes constantes de* $F_{H\Sigma}(X)$, *y como la composición con* f *sobre las componentes dependientes de* X.

La imagen de un conjunto por el funtor asociado a una signatura es un producto indexado por las operaciones de la signatura.

Si $F, G\colon Set \to Set$ son funtores, el *funtor producto* $F \times G\colon Set \to Set$ *de* F *y* G está definido, para cada conjunto X, por $(F \times G)(X) := F(X) \times G(X)$, y de manera natural sobre las aplicaciones.

El funtor $F_{H\Sigma}$ resulta ser un producto, estando cada una de sus componentes asociada a una operación de la signatura. Así, vemos a $F_{H\Sigma}$ como el funtor producto $\prod_{\sigma\in H\Sigma} F^\sigma_{H\Sigma}$ de forma que cada $F^\sigma_{H\Sigma}$ denota al funtor correspondiente a la operación σ. Es decir, si X es un conjunto, para cada deconstructor $(\sigma\colon h\ \omega \to v) \in H\Sigma$, $F^\sigma_{H\Sigma}(X) = D_v^{D_\omega}$ y para cada constructor $(\sigma\colon h\ \omega \to h) \in H\Sigma$, $F^\sigma_{H\Sigma}(X) = X^{D_\omega}$. Cada operación de la signatura aporta un factor a la definición del funtor. Los deconstructores aportan un factor constante mientras que éste es variable en el caso de las operaciones de actualización (único tipo de constructores que puede poseer $H\Sigma$). Por tanto, es claro que si la signatura es deconstructora, el funtor asociado es constante.

Observar que para las operaciones con un único argumento (por tanto, el argumento es h, y ω es la secuencia vacía), el factor aportado a la definición del funtor es el propio conjunto D_v si $\sigma\colon h \to v$ es un deconstructor, y, el propio conjunto al que se le está aplicando el funtor si $\sigma\colon h \to h$ es una operación de actualización.

Definición 2.6.4 *Sea* $H\Sigma$ *una signatura oculta sobre* D *y* $F_{H\Sigma}$ *el funtor asociado a* $H\Sigma$. *Llamaremos* categoría de coálgebras asociada a $H\Sigma$ *a la categoría* $CoAlg(F_{H\Sigma})$.

Como ejemplo, vamos a dar la expresión de los funtores asociados a las dos signaturas ocultas introducidas en 2.5.2.

Ejemplo 2.6.5

1. Volviendo a la signatura oculta $H\Sigma_1$ sobre el álgebra de datos con soportes $\mathbb{Z}, \mathbb{N}$, y $\mathcal{B}$ para los géneros visibles int, nat y $bool$ respectivamente, introducida en el ejemplo 2.5.2, el funtor $F_{H\Sigma_1}$ asociado a la signatura oculta $H\Sigma_1$ está definido, para cada conjunto X, del siguiente modo:

$$F_{H\Sigma_1}(X) := \mathbb{Z} \times \mathbb{N} \times \mathcal{B}^{\mathbb{Z}} \times \mathcal{B}^{\mathbb{N}} \times X^{\mathbb{Z}\times\mathbb{N}} \times X^{\mathbb{N}} \times X$$

y, para cada $g\colon X \to Y$ aplicación, $F_{H\Sigma_1}(g) = \prod_{\sigma\in H\Omega_1} F_{H\Sigma_1}(g)^\sigma$ donde:

- $F^{abc}_{H\Sigma_1}(g) := id_{\mathbb{Z}}$; $F^{ord}_{H\Sigma_1}(g) := id_{\mathbb{N}}$; $F^{esx?}_{H\Sigma_1}(g) := id_{\mathcal{B}^{\mathbb{Z}}}$; $F^{esy?}_{H\Sigma_1}(g) := id_{\mathcal{B}^{\mathbb{N}}}$;
- $F^{move}_{H\Sigma_1}(g)\colon X^{\mathbb{Z}\times\mathbb{N}} \to Y^{\mathbb{Z}\times\mathbb{N}}$ dada, para cada $f \in X^{\mathbb{Z}\times\mathbb{N}}$ por $F^{move}_{H\Sigma_1}(g)(f) := g \circ f$;
- $F^{esc}_{H\Sigma_1}(g)\colon X^{\mathbb{N}} \to Y^{\mathbb{N}}$ definida, para cada $f \in X^{\mathbb{N}}$ por $F^{esc}_{H\Sigma_1}(g)(f) := g \circ f$; y,
- $F^{sim}_{H\Sigma_1}(g)\colon X \to Y$ definida como la propia aplicación g.

2. La signatura oculta $H\Sigma_2$ sobre $(\{v\}, \mathbb{Z})$, introducida en el ejemplo 2.5.2 para trabajar con grupos de enteros, es deconstructora, por lo que $F_{H\Sigma_2}$, funtor asociado a ella, es constante sobre un conjunto, concretamente, sobre el conjunto $\mathbb{Z}^{\mathbb{Z}\times\mathbb{Z}} \times \mathbb{Z}^{\mathbb{Z}} \times \mathbb{Z}$.

□

Sea $F_{H\Sigma}$ el funtor asociado a una signatura oculta $H\Sigma = \langle VG \sqcup \{h\}, H\Omega\rangle$ sobre D. La descomposición del funtor $F_{H\Sigma}$ como producto $\prod_{\sigma\in H\Omega} F^{\sigma}_{H\Sigma}$ permite dar otra descripción de las $F_{H\Sigma}$-coálgebras. Cada $F_{H\Sigma}$-coálgebra $\alpha_X \colon X \to F_{H\Sigma}(X)$ puede identificarse con el conjunto X junto con una colección de aplicaciones $\{\alpha^{\sigma}_X \colon X \to F^{\sigma}_{H\Sigma}(X)\}_{(\sigma\colon h\ \omega\to g)\in H\Omega}$, con $g \in VG \sqcup \{h\}$, siendo cada aplicación $\alpha^{\sigma}_X \colon X \to F^{\sigma}_{H\Sigma}(X)$, la proyección de la aplicación α_X en la componente σ. Además, usando la adjunción entre el producto y la exponenciación en la categoría de conjuntos, si $\sigma\colon h\ \omega \to v$ es un deconstructor, en lugar de la aplicación $\alpha^{\sigma}_X \colon X \to D_v^{D_\omega}$ podemos considerar su adjunta $\sigma^{\alpha}_X \colon X \times D_\omega \to D_v$. Análogamente, si $\sigma\colon h\ \omega \to h$ es un constructor, la aplicación $\alpha^{\sigma}_X \colon X \to X^{D_\omega}$ determina una única aplicación $\sigma^{\alpha}_X \colon X \times D_\omega \to X$. Por tanto, podemos identificar cada $F_{H\Sigma}$-coálgebra (X, α_X) con la siguiente estructura:

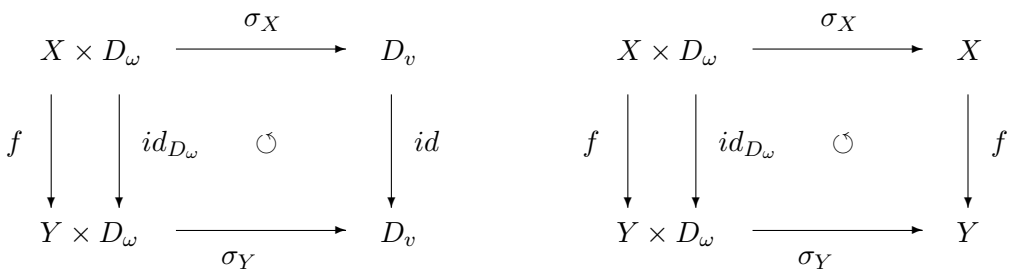
$$\Big(X, \{\sigma^{\alpha}_X \colon X \times D_\omega \to D_v\}_{(\sigma\colon h\ \omega\to v)\in H\Sigma}, \{\sigma^{\alpha}_X \colon X \times D_\omega \to X\}_{(\sigma\colon h\ \omega\to h)\in H\Sigma}\Big)$$

De esta forma, cada $F_{H\Sigma}$-coálgebra determina una $H\Sigma$-álgebra oculta, sin más que tomar el conjunto X como soporte para el género oculto h y la familia de aplicaciones como la interpretación de las operaciones en el álgebra oculta. Evidentemente, el recíproco también es cierto, es decir, cada $H\Sigma$-álgebra oculta determina una $F_{H\Sigma}$-coálgebra. Por tanto, las categorías $CoAlg(F_{H\Sigma})$ y $HAlg^{D}(H\Sigma)$ poseen los mismos objetos.

Ejemplo 2.6.6 Volviendo a las signaturas ocultas $H\Sigma_1$ y $H\Sigma_2$ introducidas en el ejemplo 2.5.2, los conjuntos asociados al género oculto h junto con la interpretación de las operaciones de las álgebras ocultas dadas en el ejemplo 2.5.6, definen coálgebras para los funtores asociados a las signaturas ocultas y cuya expresión mostramos en el ejemplo 2.6.5. □

La descripción de una $F_{H\Sigma}$-coálgebra como un conjunto junto con una colección de aplicaciones, permite caracterizar los morfismos entre $F_{H\Sigma}$-coálgebras del siguiente modo.

Teorema 2.6.7 *Sean* $\big(X, \{\sigma_X \colon X \times D_\omega \to D_v\}_{(\sigma\colon h\ \omega\to v)\in H\Sigma}, \{\sigma_X \colon X \times D_\omega \to X\}_{(\sigma\colon h\ \omega\to h)\in H\Sigma}\big)$ *e* $\big(Y, \{\sigma_Y \colon Y \times D_\omega \to D_v\}_{(\sigma\colon h\ \omega\to v)\in H\Sigma}, \{\sigma_Y \colon Y \times D_\omega \to Y\}_{(\sigma\colon h\ \omega\to h)\in H\Sigma}\big)$ *$F_{H\Sigma}$-coálgebras. Una aplicación $f\colon X \to Y$ es un morfismo entre las coálgebras si y solo si para cada deconstructor $(\sigma\colon h\ \omega \to v) \in H\Sigma$ se verifica que $\sigma_X = \sigma_Y \circ (f, id_{D_\omega})$ y, para cada constructor $(\sigma\colon h\ \omega \to h) \in H\Sigma$ se tiene que $f\circ\sigma_X = \sigma_Y\circ(f, id_{D_\omega})$; es decir, hace conmutativos los siguientes diagramas:*



Por tanto, cada morfismo entre $H\Sigma$-álgebras ocultas, define un morfismo entre las $F_{H\Sigma}$-coálgebras correspondientes (por olvido de las identidades sobre D), y, recíprocamente, completando cada morfismo entre $F_{H\Sigma}$-coálgebras con las identidades sobre D, se obtiene un morfismo entre $H\Sigma$-álgebras ocultas. Por tanto, es evidente que las categorías $CoAlg(F_{H\Sigma})$ y $HAlg^D(H\Sigma)$ pueden identificarse de manera natural. Así, cada $H\Sigma$-álgebra oculta puede interpretarse como una $F_{H\Sigma}$-coálgebra.

En el caso de una signatura Σ_{imp}, vimos que las categorías $HAlg^D(\Sigma_{imp})$ y $Alg^{D,\{\}}(\Sigma_{imp})$ pueden identificarse, por lo que cada Σ_{imp}-álgebra con soporte D puede verse como una $F_{H\Sigma}$-coálgebra. Resultado que ya mostramos en la página 65 (puesto que el funtor $F_{\Sigma_{imp}}$ coincide con el que habíamos llamado F_Σ), y que acabamos de obtener por otra vía.

2.6.3.1 Existencia de coálgebra final

Igual que en las secciones anteriores, vamos a centrarnos en estudiar la existencia de objeto final, en este caso, en la categoría de coálgebras $CoAlg(F_{H\Sigma})$ asociada a una signatura oculta $H\Sigma$ sobre D.

La identificación natural entre las categorías $CoAlg(F_{H\Sigma})$ y $HAlg^D(H\Sigma)$ junto con el teorema 2.5.12 que asegura la existencia de objeto final en $HAlg^D(H\Sigma)$, permiten asegurar la existencia de coálgebra final en $CoAlg(F_{H\Sigma})$. Ahora bien, vamos a ver cómo probar la existencia de coálgebra final sin basarnos en el resultado obtenido para categorías de álgebras ocultas. Preferimos hacerlo así porque de este modo veremos que la descripción natural obtenida para la coálgebra final coincide con la obtenida a través de la operación $()_{Set}$.

Existen trabajos, como por ejemplo [2], [8] y [9], en los que se estudia la existencia de objeto final en categorías generales de coálgebras y se muestran resultados sobre su existencia bajo determinadas propiedades del funtor que las define. Algunos de estos trabajos incluyen descripciones de la coálgebra final [8], descripciones que, en el caso general, no resultan sencillas ni intuitivas. Por este motivo, creemos conveniente no incluirlas y preferimos mostrar directamente la expresión del objeto final en un caso particular, que es suficiente para cubrir nuestro trabajo y cuya descripción es más sencilla.

En [106] y en [60], se estudia la existencia de objeto final en categorías de coálgebras asociadas a funtores del tipo de los que aparecen en la especificación de estructuras de datos. Para estos funtores asociados a signaturas, cuya descripción hemos dado explícitamente en la definición 2.6.3, resulta más sencillo asegurar la existencia de coálgebra final. En particular, en estos dos últimos trabajos se estudia la existencia de objeto final en categorías de coálgebras que provienen de funtores polinomiales. Como veremos, todos los funtores asociados a signaturas son polinomiales y, por tanto, los resultados recogidos en estos trabajos son suficientes para tratar las categorías de coálgebras que estudiamos.

Introducimos a continuación la noción de funtor polinomial que puede encontrarse, entre otros, en [106] y [60]. Previamente recogemos algunas definiciones básicas sobre funtores en Set.

Definición 2.6.8

i) Dados $F, G\colon Set \to Set$ funtores, el funtor suma $F+G\colon Set \to Set$ de F y G *viene dado, para cada conjunto X, por $(F+G)(X) := F(X) \sqcup G(X)$, y, definido de manera natural sobre las aplicaciones.*

ii) *Sea A un conjunto, se define el* funtor exponencial sobre A *como el endofuntor de Set que asocia, a cada conjunto X, el espacio de funciones X^A, y, de manera natural, asocia a cada aplicación $f\colon X \to Y$ la correspondiente de X^A en Y^A dada por la composición con la aplicación f.*

iii) *Un funtor $F\colon Set \to Set$ es* polinomial *si es un funtor construido a partir del funtor identidad, funtores constantes sobre conjuntos y funtores producto, suma o exponencial sobre conjuntos. Es decir, F es polinomial si es de la forma*

$$F(X) = \prod_{i=1}^{n} (B_i + C_i \times X)^{A_i}$$

con A_i, B_i, C_i conjuntos constantes para todo $i = 1, \ldots, n$.

Observar que los funtores constantes sobre conjuntos son un caso particular de funtores polinomiales. Notar que las definiciones recogidas en 2.6.8 son correctas porque la categoría *Set* posee coproductos, productos finitos y exponenciación.

A la vista de su descripción, es evidente que los funtores asociados a signaturas ocultas son polinomiales.

El siguiente resultado, que puede encontrarse en [8], [106] y [60], asegura la existencia de coálgebra final en categorías de coálgebras definidas por funtores polinomiales.

Teorema 2.6.9 *Si $F\colon Set \to Set$ es un funtor polinomial entonces la categoría $CoAlg(F)$ posee objeto final.*

Por tanto, las categorías $CoAlg(F_{H\Sigma})$ asociadas a signaturas ocultas poseen objeto final. Resultado ya conocido por el isomorfismo entre las categorías $CoAlg(F_{H\Sigma})$ y $HAlg^D(H\Sigma)$, pero que acabamos de obtener directamente sin pasar por las categorías de álgebras ocultas.

2.6.3.2 Descripción de la coálgebra final

La identificación natural entre las categorías $CoAlg(F_{H\Sigma})$ y $HAlg^D(H\Sigma)$ hace que la descripción dada para la $H\Sigma$-álgebra final en el apartado 2.5.3.1 sea una descripción de la coálgebra final en $CoAlg(F_{H\Sigma})$.

Sin embargo, vamos a dar otra descripción de la coálgebra final en $CoAlg(F_{H\Sigma})$, simplemente como coálgebra, sin utilizar el isomorfismo con otras categorías. Esta descripción puede encontrarse en [91] y en [60].

Sean $H\Sigma = \langle VG \sqcup \{h\}, H\Omega \rangle$ una signatura oculta y $F_{H\Sigma}$ el funtor asociado a la signatura, es decir, $F_{H\Sigma}$ viene dado, para cada conjunto X, por

$$F_{H\Sigma}(X) := \prod_{(\sigma\colon h\ \omega \to v) \in H\Sigma} D_v^{D_\omega} \times \prod_{(\sigma\colon h\ \omega \to h) \in H\Sigma} X^{D_\omega}$$

Observar que el conjunto $\prod_{(\sigma\colon h\ \omega \to h) \in H\Sigma} X^{D_\omega}$ es biyectivo con el conjunto $X^{\coprod_{(\sigma\colon h\ \omega \to h) \in H\Sigma} D_\omega}$ y, es éste último, el que vamos a utilizar para dar, a continuación, una descripción de la coálgebra final, coálgebra que denotaremos por

$$\Big(\mathbb{F}, \{\sigma_{\mathbf{F}}\colon \mathbb{F} \times D_\omega \to D_v\}_{(\sigma\colon h\ \omega \to v) \in H\Sigma}, \{\sigma_{\mathbf{F}}\colon \mathbb{F} \times D_\omega \to \mathbb{F}\}_{(\sigma\colon h\ \omega \to h) \in H\Sigma} \Big)$$

- El conjunto base de la coálgebra final viene descrito del siguiente modo:

$$\mathbb{F} = \Big(\prod_{(\sigma\colon h\ \omega \to v) \in H\Sigma} D_v^{D_\omega} \Big)^{\left(\bigsqcup_{(\delta\colon h\ \omega' \to h) \in H\Sigma} D_{\omega'}\right)^*}$$

  (Como es habitual, si X es un conjunto, X^* denota al conjunto de secuencias de elementos de X, incluyendo la secuencia vacía.) Es decir, cada $f \in \mathbb{F}$ es una tupla de funciones, una por cada deconstructor de la signatura:

$$f = \Big(f_\sigma\colon \big(\bigsqcup_{(\delta\colon h\ \omega' \to h) \in H\Sigma} D_{\omega'} \big)^* \to D_v \Big)_{(\sigma\colon h\ \omega \to v) \in H\Sigma}$$

  Para cada deconstructor $(\sigma\colon h\ \omega \to v) \in H\Sigma$, un elemento del dominio de la función asociada, es decir, un elemento de $\left(\bigsqcup_{(\delta\colon h\ \omega' \to h) \in H\Sigma} D_{\omega'}\right)^*$, es una secuencia de datos de forma que, cada dato es un elemento de un dominio $D_{\omega'}$ asociado a un constructor $(\delta\colon h\ \omega' \to h) \in H\Sigma$.

  La relación con el soporte para el género oculto en el objeto final de la categoría de álgebras ocultas descrito en el apartado 2.5.3.1 es clara. El considerar como dominio de las funciones las secuencias de elementos de $D_{\omega'}$ siendo ω' los argumentos visibles de algún constructor, corresponde con la idea de contexto como aplicación sucesiva de constructores seguida de la aplicación de un deconstructor.

- Para cada operación de $H\Sigma$ hay que dar una función.

  · Si $\sigma\colon h\ \omega \to v$ es un deconstructor, $\sigma_{\mathbf{F}}\colon \mathbb{F} \times D_\omega \to D_v$ se define, para cada $f \in \mathbb{F}$ y para cada $d_\omega \in D_\omega$ por $\sigma_{\mathbf{F}}(f, d_\omega) := f_\sigma([])(d_\omega)$.
  Si $\omega = []$, entonces $\sigma_{\mathbf{F}}(f) := f_\sigma([])$, para cada $f \in \mathbb{F}$.

  · Si $\sigma\colon h\ \omega \to h$ es un constructor, $\sigma_{\mathbf{F}}\colon \mathbb{F} \times D_\omega \to \mathbb{F}$ se define por $\sigma_{\mathbf{F}}(f, d_\omega) := (f_\delta(d_\omega \wedge -))_{(\delta\colon h\ \omega' \to v) \in H\Sigma}$, para cada $f \in \mathbb{F}$ y para cada $d_\omega \in D_\omega$, donde $\wedge$ denota la operación de concatenación de cadenas.

  La relación con el álgebra oculta final descrita en el apartado 2.5.3.1 es clara. Cada elemento del conjunto $\left(\bigsqcup_{(\delta\colon h\ \omega' \to h) \in H\Sigma} D_{\omega'}\right)^*$ está formado por una secuencia de datos cada uno de los cuales pertenece al conjunto $D_{\omega'}$ siendo ω' los argumentos visibles de algún constructor. Así, cada elemento del conjunto anterior determina una secuencia de constructores (determinada a partir de los datos que forman el elemento) que aplicándolos a los datos de la secuencia y componiendo finalmente con el deconstructor apropiado, permite obtener un contexto.

  En las especificaciones ocultas los contextos dan la secuencia de constructores seguida de un deconstructor a interpretar así como los elementos de D sobre los que se aplican. En este caso, los elementos de los dominios determinan la secuencia de constructores a aplicar y la componente en la que estamos determina el deconstructor.

En el caso de que la signatura sea deconstructora, el conjunto base para la coálgebra es el conjunto $\mathbb{F} = \prod_{(\sigma\colon h\ \omega \to v) \in H\Sigma} D_v^{D_\omega}$, por lo que la expresión de la coálgebra se reduce notablemente.

Vamos a ver esta descripción del objeto final en el caso de las categorías de coálgebras asociadas a las signaturas introducidas en el ejemplo 2.5.2.

Ejemplo 2.6.10

1. Volviendo a la signatura oculta $H\Sigma_1$ introducida en el ejemplo 2.5.2, vimos (en la sección 2.6.5) que el funtor asociado a $H\Sigma_1$ viene dado, para cada conjunto X, por:

$$F_{H\Sigma_1}(X) := \mathbb{Z} \times \mathbb{N} \times \mathcal{B}^{\mathbb{Z}} \times \mathcal{B}^{\mathbb{N}} \times X^{\mathbb{Z}\times\mathbb{N}} \times X^{\mathbb{N}} \times X$$

o equivalentemente podemos verlo como

$$F_{H\Sigma_1}(X) := \mathbb{Z} \times \mathbb{N} \times \mathcal{B}^{\mathbb{Z}} \times \mathcal{B}^{\mathbb{N}} \times X^{(\mathbb{Z}\times\mathbb{N}\sqcup\mathbb{N}\sqcup\{*\})}$$

La $F_{H\Sigma_1}$-coálgebra final podemos expresarla, según la descripción que acabamos de dar del siguiente modo:

- El conjunto viene dado por $\mathbb{F}_{H\Sigma_1} = \left(\mathbb{Z} \times \mathbb{N} \times \mathcal{B}^{\mathbb{Z}} \times \mathcal{B}^{\mathbb{N}}\right)^{(\mathbb{Z}\times\mathbb{N}\sqcup\mathbb{N}\sqcup\{*\})^*}$

  Si $f \in \mathbb{F}_{H\Sigma_1}$, entonces f es una función $f\colon (\mathbb{Z} \times \mathbb{N} \sqcup \mathbb{N} \sqcup \{*\})^* \to \mathbb{Z} \times \mathbb{N} \times \mathcal{B}^{\mathbb{Z}} \times \mathcal{B}^{\mathbb{N}}$, y podemos descomponerla del siguiente modo: $f = (f_{abc}, f_{ord}, f_{esx?}, f_{esy?})$ con

  · $f_{abc}\colon (\mathbb{Z} \times \mathbb{N} \sqcup \mathbb{N} \sqcup \{*\})^* \to \mathbb{Z}$
  · $f_{ord}\colon (\mathbb{Z} \times \mathbb{N} \sqcup \mathbb{N} \sqcup \{*\})^* \to \mathbb{N}$
  · $f_{esx?}\colon (\mathbb{Z} \times \mathbb{N} \sqcup \mathbb{N} \sqcup \{*\})^* \to \mathcal{B}^{\mathbb{Z}}$
  · $f_{esy?}\colon (\mathbb{Z} \times \mathbb{N} \sqcup \mathbb{N} \sqcup \{*\})^* \to \mathcal{B}^{\mathbb{N}}$

  La relación entre el conjunto $\mathbb{F}_{H\Sigma_1}$ y el conjunto $F_h^{H\Sigma_1} = \mathbb{Z}^{\mathcal{C}_{H\Sigma_1}[z_h]_{int}} \times \mathbb{N}^{\mathcal{C}_{H\Sigma_1}[z_h]_{nat}} \times \mathcal{B}^{\mathcal{C}_{H\Sigma_1}[z_h]_{bool}}$, tomado como soporte para el álgebra oculta final en el ejemplo 2.5.14, es clara. Cada $f = (f_{abc}, f_{ord}, f_{esx?}, f_{esy?}) \in \mathbb{F}_{H\Sigma_1}$, define un elemento $(f_{int}, f_{nat}, f_{bool})$ de $F_h^{H\Sigma_1}$: dado un contexto de género *int* se considera la secuencia formada por sus argumentos visibles, cada uno de ellos correspondiente a un constructor, y se le aplica la función f_{abc}, el resultado es el utilizado para definir f_{int}; análogamente para los contextos de género *nat*; y, para los de género *bool*, si el deconstructor aplicado es *esx*?, se aplica la función $f_{esx?}$ mientras que si el deconstructor es *esy*? se aplica $f_{esy?}$. Inversamente, dado un elemento de $(\mathbb{Z} \times \mathbb{N} \sqcup \mathbb{N} \sqcup \{*\})^*$, se considera la secuencia de constructores asociada a la secuencia de datos que forman el elemento y se aplica en último lugar el deconstructor correspondiente a la función que se pretende definir, lo que da lugar a un contexto de género el género resultado del deconstructor y al que se le aplica la función correspondiente.

- Para cada operación de $H\Sigma_1$ damos una aplicación, de forma que para cada $f = (f_{abc}, f_{ord}, f_{esx?}, f_{esy?}) \in \mathbb{F}_{H\Sigma_1}$, para cada $z \in \mathbb{Z}$ y para cada $n \in \mathbb{N}$, se definen:

  · $abc_{\mathbf{F}_{H\Sigma_1}} : \mathbb{F}_{H\Sigma_1} \to \mathbb{Z}$ dada por $abc_{\mathbf{F}_{H\Sigma_1}}(f) := f_{abc}([])$;
  · $ord_{\mathbf{F}_{H\Sigma_1}} : \mathbb{F}_{H\Sigma_1} \to \mathbb{N}$ dada por $ord_{\mathbf{F}_{H\Sigma_1}}(f) := f_{ord}([])$;
  · $esx?_{\mathbf{F}_{H\Sigma_1}} : \mathbb{F}_{H\Sigma_1} \times \mathbb{Z} \to \mathcal{B}$ dada por $esx?_{\mathbf{F}_{H\Sigma_1}}(f, z) := f_{esx?}([])(z)$. Como $f_{esx?}([]) \in \mathcal{B}^{\mathbb{Z}}$, $esx?_{\mathbf{F}_{H\Sigma_1}}$ está bien definida;
  · $esy?_{\mathbf{F}_{H\Sigma_1}} : \mathbb{F}_{H\Sigma_1} \times \mathbb{N} \to \mathcal{B}$ dada por $esy?_{\mathbf{F}_{H\Sigma_1}}(f, n) := f_{esy?}([])(n)$;
  · $move_{\mathbf{F}_{H\Sigma_1}} : \mathbb{F}_{H\Sigma_1} \times \mathbb{Z} \times \mathbb{N} \to \mathbb{F}_{H\Sigma_1}$ definida por $move_{\mathbf{F}_{H\Sigma_1}}(f, z, n) := f((z, n) \wedge -)$, siendo $f((z, n) \wedge -) = \big(f_{abc}((z, n) \wedge -),\ f_{ord}((z, n) \wedge -),\ f_{esx?}((z, n) \wedge -),\ f_{esy?}((z, n) \wedge -)\big)$, donde $\wedge$ denota la operación de concatenación de cadenas;

- $esc_{\mathbf{F}_{H\Sigma_1}} : \mathbb{F}_{H\Sigma_1} \times \mathbb{N} \to \mathbb{F}_{H\Sigma_1}$ dada por $esc_{\mathbf{F}_{H\Sigma_1}}(f, n) := f(n \wedge -)$, siendo $f(n \wedge -) = \big(f_{abc}(n \wedge -), f_{ord}(n \wedge -), f_{esx?}(n \wedge -), f_{esy?}(n \wedge -)\big)$; y, finalmente,
- $sim_{\mathbf{F}_{H\Sigma_1}} : \mathbb{F}_{H\Sigma_1} \to \mathbb{F}_{H\Sigma_1}$ dada por $sim_{\mathbf{F}_{H\Sigma_1}}(f) := f(* \wedge -)$, siendo $f(* \wedge -) = \big(f_{abc}(* \wedge -), f_{ord}(* \wedge -), f_{esx?}(* \wedge -), f_{esy?}(* \wedge -)\big)$.

2. En el caso de la signatura $H\Sigma_2$ sobre $\mathbb{Z}$, tal y como vimos en el ejemplo 2.6.5, el funtor asociado es el funtor constante sobre el conjunto $\mathbb{Z}^{\mathbb{Z}\times\mathbb{Z}} \times \mathbb{Z}^{\mathbb{Z}} \times \mathbb{Z}$. La descripción de la coálgebra final es más sencilla que en el ejemplo anterior por ser todas las operaciones deconstructoras.

- $\mathbb{F}_{H\Sigma_2} = \mathbb{Z}^{\mathbb{Z}\times\mathbb{Z}} \times \mathbb{Z}^{\mathbb{Z}} \times \mathbb{Z}$

  Por tanto, cada $f \in \mathbb{F}_{H\Sigma_2}$ podemos describirla como $f = (f_{prd\_grp}, f_{inv\_grp}, z_0)$ siendo

  - $f_{prd\_grp} \colon \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$
  - $f_{inv\_grp} \colon \mathbb{Z} \to \mathbb{Z}$
  - $z_0 \in \mathbb{Z}$

  La relación entre el conjunto $\mathbb{F}_{H\Sigma_2}$ y el soporte para el género oculto del álgebra final, dada en el ejemplo 2.5.14, es clara ya que los contextos que provienen del deconstructor *prd_grp* permiten definir un elemento de $\mathbb{Z}^{\mathbb{Z}\times\mathbb{Z}}$, los que provienen de *inv_grp* uno de $\mathbb{Z}^{\mathbb{Z}}$ y el único contexto que proviene de *unt_grp* permite determinar z_0.

- La familia de aplicaciones viene dada, para cada $f \in \mathbb{F}_{H\Sigma_2}$ y para cada $z_1, z_2 \in \mathbb{Z}$ por:

  - $prd\_grp_{\mathbf{F}_{H\Sigma_2}} : \mathbb{F}_{H\Sigma_2} \times \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ definida por $prd\_grp_{\mathbf{F}_{H\Sigma_2}}(f, z_1, z_2) := f_{prd\_grp}(z_1, z_2)$;
  - $inv\_grp_{\mathbf{F}_{H\Sigma_2}} : \mathbb{F}_{H\Sigma_2} \times \mathbb{Z} \to \mathbb{Z}$ dada por $inv\_grp_{\mathbf{F}_{H\Sigma_2}}(f, z_1) := f_{inv\_grp}(z_1)$;
  - $unt\_grp_{\mathbf{F}_{H\Sigma_2}} : \mathbb{F}_{H\Sigma_2} \to \mathbb{Z}$ dada por $unt\_grp_{\mathbf{F}_{H\Sigma_2}}(f) := z_0$.

  En este caso, es fácil ver que la función definida sobre contextos se sustituye por la función que define el conjunto de contextos al que pertenece.

  Observar que, precisamente ésta es la descripción del objeto final en $Alg^{D,\{\}}(\mathtt{GRP}_{imp})$ que dimos en el ejemplo 2.4.3.

□

2.6.3.3 Expresión del morfismo final

Además, podemos dar la expresión del morfismo final para cualquier $F_{H\Sigma}$-coálgebra. Sea $\big(A, \{\sigma_A \colon A \times D_\omega \to D_v\}_{(\sigma\colon h\ \omega\to v)\in H\Sigma}, \{\sigma_A \colon A \times D_\omega \to A\}_{(\sigma\colon h\ \omega\to h)\in H\Sigma}\big)$ una $F_{H\Sigma}$-coálgebra y sea $\big(\mathbb{F}, \{\sigma_{\mathbf{F}} \colon \mathbb{F} \times D_\omega \to D_v\}_{(\sigma\colon h\ \omega\to v)\in H\Sigma}, \{\sigma_{\mathbf{F}} \colon \mathbb{F} \times D_\omega \to \mathbb{F}\}_{(\sigma\colon h\ \omega\to h)\in H\Sigma}\big)$ la $F_{H\Sigma}$-coálgebra final.

El morfismo final viene dado por la aplicación $g \colon A \to \mathbb{F}$ definida, para cada $a \in A$, por $g(a) := \big(g(a)_\sigma\big)_{(\sigma\colon h\ \omega\to v)\in H\Sigma}$, siendo, para cada deconstructor $(\sigma \colon h\ \omega \to v) \in H\Sigma$, $g(a)_\sigma \colon (\bigsqcup_{(\sigma\colon h\ \omega'\to h)\in H\Sigma} D_{\omega'})^* \to D_v^{D_\omega}$ la aplicación definida como sigue:

- $g(a)_\sigma([])(d_\omega) := \sigma_A(a, d_\omega)$, para todo $d_\omega \in D_\omega$. Es decir, para los elementos de D_ω se considera directamente la interpretación de la operación de A con a fijo; y,

· si $x = x_1 \dots x_n \in (\bigsqcup_{(\sigma\colon h\ \omega' \to h) \in H\Sigma} D_{\omega'})^*$ con $x_i \in \bigsqcup_{(\sigma\colon h\ \omega' \to h) \in H\Sigma} D_{\omega'}$ para todo $i = 1, \dots, n$, y $d_\omega \in D_\omega$, se define

$$g(a)_\sigma(x)(d_\omega) := \sigma_A(\delta_{n_A}(\delta_{n-1_A}(\dots(\delta_{1_A}(a, x_1), x_2), \dots), x_n), d_\omega)$$

siendo, para cada $i = 1, \dots, n$, $(\delta_i\colon h\ \omega' \to h) \in H\Sigma$ la única operación de $H\Sigma$ tal que $x_i \in D_{\omega'}$.

Es decir, se aplican los constructores correspondientes a los datos de la secuencia x y, finalmente, se aplica el deconstructor.

Es claro que así definido g hace conmutativos todos los diagramas correspondientes a las operaciones y, por tanto, es un morfismo de coálgebras.

Veamos la expresión concreta en el caso de las signaturas $H\Sigma_1$ y $H\Sigma_2$ cuyos objetos finales hemos descrito en el ejemplo 2.6.10.

Ejemplo 2.6.11

1. Si $(A, \{abc_A, ord_A, esx?_A, esy?_A\}, \{move_A, esc_A, sim_A\})$ es una $F_{H\Sigma_1}$-coálgebra, el morfismo final $g\colon A \to \mathbb{F}_{H\Sigma_1}$ viene dado, para cada $a \in A$ por:

$$g(a) := \big(g(a)_{abc}, g(a)_{ord}, g(a)_{esx?}, g(a)_{esy?}\big) \text{ con}$$

· $g(a)_{abc}\colon (\mathbb{Z} \times \mathbb{N} \sqcup \mathbb{N} \sqcup \{*\})^* \to \mathbb{Z}$ aplicación definida, para cada $x \in (\mathbb{Z} \times \mathbb{N} \sqcup \mathbb{N} \sqcup \{*\})^*$, siendo $x = x_1 x_2 \dots x_n$ con $x_i \in \{\mathbb{Z} \times \mathbb{N} \sqcup \mathbb{N} \sqcup \{*\}\}$ para todo $i = 1, \dots, n$,

$$g(a)_{abc}(x) := abc_A(met_n(\dots(met_1(a, x_1), \dots), x_n))$$

donde, para cada $i \in \{1, \dots n\}$,

$$met_i = \begin{cases} move_A & \text{si } x_i \in \mathbb{Z} \times \mathbb{N} \\ esc_A & \text{si } x_i \in \mathbb{N} \\ sim_A & \text{si } x_i \in \{*\} \end{cases}$$

Análogamente se define $g(a)_{ord}$.

· $g(a)_{esx?}\colon (\mathbb{Z} \times \mathbb{N} \sqcup \mathbb{N} \sqcup \{*\})^* \to \mathcal{B}^{\mathbb{Z}}$ definida, para cada $z \in \mathbb{Z}$ y para cada $x \in (\mathbb{Z} \times \mathbb{N} \sqcup \mathbb{N} \sqcup \{*\})^*$ con $x = x_1 x_2 \dots x_n$ y $x_i \in \{\mathbb{Z} \times \mathbb{N} \sqcup \mathbb{N} \sqcup \{*\}\}$ para todo $i = 1, \dots, n$,

$$g(a)_{esx?}(x)(z) := esx?_A(met_n(\dots(met_1(a, x_1), \dots), x_n), z)$$

siendo, para cada $i \in \{1, \dots n\}$, met_i definido como en las operaciones anteriores.

Análogamente se define $g(a)_{esy?}$.

2. Si $(B, \{prd\_grp_B, inv\_grp_B, unt\_grp_B\})$ es una $F_{H\Sigma_2}$-coálgebra, el morfismo final $g\colon B \to \big(\mathbb{Z}^{\mathbb{Z}\times\mathbb{Z}} \times \mathbb{Z}^{\mathbb{Z}} \times \mathbb{Z}\big)$ viene dado, para cada $b \in B$, por

$$g(b) := \big(g(b)_{prd\_grp}, g(b)_{inv\_grp}, unt\_grp_B(b)\big)$$

siendo, para todo $z_1, z_2 \in \mathbb{Z}$,

· $g(b)_{prd\_grp}\colon \mathbb{Z}\times\mathbb{Z}\to\mathbb{Z}$ la aplicación dada por $g(b)_{prd\_grp}(z_1,z_2) := prd\_grp_B(b,z_1,z_2)$, y

· $g(b)_{inv\_grp}\colon \mathbb{Z}\to\mathbb{Z}$ la definida por $g(b)_{inv\_grp}(z_1) := inv\_grp_B(b,z_1)$

□

En el caso de las signaturas anteriores, observando las expresiones de los morfismos finales entre álgebras ocultas dadas en el ejemplo 2.5.15, es evidente que se parecen mucho a las dadas en el ejemplo anterior como morfismos entre coálgebras.

2.6.4 Categorías de coálgebras asociadas a las signaturas Σ_{imp}

Volvemos ahora a las signaturas Σ_{imp}. Vamos a recopilar las relaciones que hemos ido introduciendo entre las Σ_{imp}-álgebras y las distintas aproximaciones con las que hemos ido trabajando.

Sea $\Sigma = \langle G,\Omega\rangle$ una signatura y sea $\Sigma_{imp} = \langle G \sqcup \{imp_\Sigma\}, \Omega_{imp}\rangle$ la signatura obtenida aplicando la operación $()_{imp}$ a la signatura Σ. Por definición de la signatura Σ_{imp}, $\Omega_{imp} = \{imp\_\sigma\colon imp_\Sigma\ \omega\to g\}_{(\sigma\colon \omega\to g)\in\Omega}$. Por tanto, el género distinguido imp_Σ aparece solo una vez y siempre como primer argumento en todas las operaciones de Σ_{imp}. Además, como todas las operaciones son observadores, el funtor asociado a Σ_{imp} resultará ser un funtor constante.

Por tanto, fijando un G-conjunto D, podemos definir el funtor $F_{\Sigma^D_{imp}}$ asociado a Σ_{imp} y a D, como el funtor constante sobre el conjunto $\prod_{(imp\_\sigma\colon imp_\Sigma\ \omega\to g)\in\Sigma_{imp}} D_g^{D_\omega}$ o lo que es lo mismo sobre el conjunto $\prod_{(\sigma\colon \omega\to g)\in\Sigma} D_g^{D_\omega}$.

Observar que el funtor $F_{\Sigma^D_{imp}}$ coincide con el que en la sección 2.6.2 habíamos denotado por F_Σ (funtor asociado a Σ y a D).

Además, la categoría de coálgebras $CoAlg(F_{\Sigma^D_{imp}})$ posee objeto final, siendo una descripción de este objeto la mostrada en el apartado 2.6.3.2. En este caso particular el objeto final puede definirse del siguiente modo:

- Como conjunto:
$$\mathbb{F}_{\Sigma^D_{imp}} = \prod_{(\sigma\colon \omega\to g)\in\Sigma} D_g^{D_\omega}$$
- Para cada operación $(\sigma\colon \omega\to g)\in\Sigma$, se define la aplicación $\sigma_{\mathbf{F}_{\Sigma^D_{imp}}} : \mathbb{F}_{\Sigma^D_{imp}}\times D_\omega \to D_g$ dada por $\sigma_{\mathbf{F}_{\Sigma^D_{imp}}}(f,d_\omega) := f_\sigma(d_\omega)$, para todo $f\in\mathbb{F}_{\Sigma^D_{imp}}$ y para todo $d_\omega\in D_\omega$.

La descripción anterior es exactamente la que dimos para el objeto final en la categoría $Alg^{D,\{\}}(\Sigma_{imp})$.

Por tanto, recopilando todo lo visto hasta ahora para las signaturas Σ_{imp} tenemos:

- Habíamos denotado por $Alg^{D,\{\}}(\Sigma_{imp})$ a la categoría de Σ_{imp}-álgebras con soporte D para los géneros de Σ y con morfismos los que son la identidad sobre D. En 2.6.2 recogimos el isomorfismo entre las categorías $CoAlg(F_{\Sigma^D_{imp}})$ y $Alg^{D,\{\}}(\Sigma_{imp})$, lo que permite interpretar cada Σ_{imp}-álgebra con soporte D como una $F_{\Sigma^D_{imp}}$-coálgebra.

Además, en el teorema 2.4.6 vimos que el objeto final de $Alg^{D,\{\}}(\Sigma_{imp})$ es el objeto $\mathbb{1}^D$, una de cuyas descripciones, según vimos en la sección 2.4.2, coincide con la que acabamos de mostrar para la coálgebra final en $CoAlg(F_{\Sigma^D_{imp}})$.

- Observar que el funtor $F_{\Sigma^D_{imp}}$ (o F_Σ como lo habíamos denotado previamente) es equivalente al funtor $F_{Alg^D(\Sigma)}$ definido como el funtor constante sobre el conjunto $Alg^D(\Sigma)$, por lo que las categorías $CoAlg(F_{Alg^D(\Sigma)})$ y $CoAlg(F_{\Sigma^D_{imp}})$ son isomorfas.

- La identificación entre la categoría de álgebras ocultas para una signatura oculta y la categoría de coálgebras definida por el funtor asociado, vista en la página 69, aplicada a la signatura Σ_{imp}, permite deducir que las categorías $HAlg^D(\Sigma_{imp})$ y $CoAlg(F_{\Sigma_{imp}})$ son isomorfas.

Por tanto, las categorías $Alg^{D,\{\}}(\Sigma_{imp})$, $CoAlg(F_{\Sigma^D_{imp}})$, $CoAlg(F_{Alg^D(\Sigma)})$ y $HAlg^D(\Sigma_{imp})$ son isomorfas. Más aún, hemos visto que existen isomorfismos entre ellas que son meras identificaciones. En esencia, se dispone de cuatro expresiones para trabajar con familias de Σ-álgebras, teniendo además, distintas descripciones para el objeto final.

2.7 Tipos Abstractos de Datos (en el marco total)

En la sección 2.2.1 hemos definido una operación entre signaturas que asocia, a cada signatura Σ, otra signatura Σ_{imp} y que, según hemos visto, puede interpretarse como la signatura para especificar determinadas familias de Σ-álgebras. Posteriormente hemos establecido relaciones entre las categorías $Alg(\Sigma)$ y $Alg(\Sigma_{imp})$. Ahora bien, teniendo en cuenta que nuestro interés es formalizar estructuras de datos utilizadas para codificar modelos determinados de antemano, debemos considerar que, en general, el punto de partida no son todas las álgebras para una signatura, sino un subconjunto propio de ellas. Por ello, generalmente no interesa trabajar con toda la categoría de Σ-álgebras, sino que resulta más natural restringirnos a aquellas álgebras que pretendemos codificar (lo habitual es que, en la práctica, las estructuras de datos usadas codifiquen más álgebras que las de partida). De este modo aparece la noción de *Tipo Abstracto de Datos*, TAD en lo que sigue. Esta noción es una de las utilizadas habitualmente para la formalización de tipos o estructuras de datos y en la literatura pueden encontrarse múltiples variantes de ella (por ejemplo, en [40], [72], [54] y [19]; [6] recoge y desarrolla algunas de las distintas variantes). En estas variantes, lo habitual es considerar una signatura como parte sintáctica del TAD y añadir algo que permita dotarle de significado. Diferentes posibilidades a la hora de establecer la semántica dan lugar a diferentes nociones de TAD. En [92] se hace un repaso a algunas de ellas. En la especificación algebraica clásica, la noción de TAD está relacionada con las signaturas con constructores y para las que la semántica inicial no es trivial, sino que aporta información. En esta línea trabaja por ejemplo [40]. A continuación introducimos la que nosotros adoptamos, noción que está muy cercana a la utilizada, por ejemplo, en [72], y que no es axiomática sino basada en modelos (ver definición en preliminares, página 27).

Definición 2.7.1 *Un* Tipo Abstracto de Datos (TAD) *es un par* $\mathcal{T} = \langle \Sigma, \mathcal{C}(\Sigma) \rangle$ *formado por una signatura* Σ *y una subcategoría* $\mathcal{C}(\Sigma)$ *de* $Alg(\Sigma)$*, cerrada por isomorfismos.*

El incluir en la definición anterior la condición de "cerrada por isomorfismos" es porque, teóricamente, en el modelado de un tipo se trabaja "salvo isomorfismo". Sin embargo, en

la práctica, a veces interesa ampliar un poco más la definición anterior, considerando subcategorías cerradas por isomorfismos no directamente como subcategorías de $Alg(\Sigma)$, sino de alguna subcategoría suya propia.

Veamos algunos ejemplos de TADs.

Ejemplo 2.7.2

1. Para especificar un tipo de datos que permita trabajar con grupos lo natural es considerar la signatura GRP introducida en el ejemplo 2.2.1, y trabajar con la subcategoría plena de GRP-álgebras con objetos los grupos cuyo conjunto subyacente es un subconjunto de $\mathcal{U}$. Denotaremos por $\mathcal{C}(\mathtt{GRP})$ a la subcategoría anterior. El TAD $\mathcal{T}_{\mathtt{GRP}} = \langle \mathtt{GRP}, \mathcal{C}(\mathtt{GRP})\rangle$ modela a los grupos (sobre subconjuntos de $\mathcal{U}$).

2. Vamos a considerar una signatura, que denominamos ANLL, y que nos permitirá modelar la estructura anillo. La signatura ANLL posee un único género g y las operaciones son las siguientes:

$$\begin{array}{lcccc} sum & : & g\ g & \to & g \\ opto & : & g & \to & g \\ ntr & : & & \to & g \\ prd & : & g\ g & \to & g \\ unt & : & & \to & g \end{array}$$

   Sea $\mathcal{C}(\mathtt{ANLL})$ subcategoría plena de $Alg(\mathtt{ANLL})$ cuyos objetos son los anillos sobre subconjuntos de $\mathcal{U}$. El TAD $\mathcal{T}_{\mathtt{ANLL}} = \langle \mathtt{ANLL}, \mathcal{C}(\mathtt{ANLL})\rangle$ permite trabajar con los anillos como estructuras algebraicas.

□

Nuestro objetivo en esta sección es mostrar que la operación $()_{imp}$ definida sobre signaturas puede extenderse a TADs. Para ello, deberemos definir a partir de una subcategoría de $Alg(\Sigma)$, posiblemente propia, una subcategoría de $Alg(\Sigma_{imp})$.

En el resto de la sección supondremos que $\Sigma = \langle G, \Omega\rangle$ es una signatura y que $\mathcal{T} = \langle \Sigma, \mathcal{C}(\Sigma)\rangle$ es un TAD. Supondremos también que los soportes de las Σ-álgebras de $\mathcal{C}(\Sigma)$ están elegidos en $\mathcal{U}$, universo algebraico, para así garantizar que $\mathcal{C}(\Sigma)$ sea una categoría pequeña.

2.7.1 Obtención de una categoría de Σ_{imp}-álgebras a partir de una categoría de Σ-álgebras

De modo natural, cada subcategoría $\mathcal{C}(\Sigma)$ de $Alg(\Sigma)$ permite definir una subcategoría de $Alg(\Sigma_{imp})$, que denotamos por $\mathcal{C}(\Sigma_{imp})$, y que definimos del siguiente modo:

- Los objetos son las Σ_{imp}-álgebras $A = \big\langle A_{imp_\Sigma}, (A_g)_{g\in G}, \{imp\_\sigma_A \colon A_{imp_\Sigma} \times A_\omega \to A_v\}_{(\sigma\colon \omega\to v)\in\Sigma}\big\rangle$ tales que, para todo $a \in A_{imp_\Sigma}$, la Σ-álgebra $A_a = \big\langle (A_g)_{g\in G}, \{imp\_\sigma_A(a,-)\colon A_\omega \to A_v\}_{(\sigma\colon \omega\to v)\in\Sigma}\big\rangle$ está en $\mathcal{C}(\Sigma)$.

- Si A y B son dos objetos de $\mathcal{C}(\Sigma_{imp})$, un Σ_{imp}-morfismo $f\colon A \to B$, $f = (f_{imp_\Sigma}, (f_g)_{g\in G})$, es un morfismo de $\mathcal{C}(\Sigma_{imp})$ si, para todo $a \in A_{imp_\Sigma}$, $(f_g)_{g\in G}\colon A_a \to B_{f_{imp_\Sigma}(a)}$ es un morfismo de $\mathcal{C}(\Sigma)$.

Así definida $\mathcal{C}(\Sigma_{imp})$ es categoría, y es cerrada por isomorfismos por serlo $\mathcal{C}(\Sigma)$.

Cada Σ_{imp}-álgebra de la categoría $\mathcal{C}(\Sigma_{imp})$ puede interpretarse como una familia de Σ-álgebras de la categoría $\mathcal{C}(\Sigma)$. Esta interpretación corresponde a restringir el funtor inclusión $I^{imp}\colon Alg(\Sigma_{imp}) \to Alg(\Sigma)_{Set}$ definido en la sección 2.4. Más concretamente, consideramos el funtor $I^{imp}_{\mathcal{C}(\Sigma_{imp})}\colon \mathcal{C}(\Sigma_{imp}) \to \mathcal{C}(\Sigma)_{Set}$, definido como la restricción del funtor I^{imp} a la categoría $\mathcal{C}(\Sigma_{imp})$. Observar que también ha cambiado la categoría de llegada, siendo correcta la definición del funtor, ya que la imagen por el funtor I^{imp} de cada Σ_{imp}-álgebra A cae dentro de $\mathcal{C}(\Sigma)_{Set}$.

El funtor $I^{imp}_{\mathcal{C}(\Sigma_{imp})}$ es fiel e inyectivo sobre objetos. Por contra, notar que este funtor solo cubre aquellos objetos de $\mathcal{C}(\Sigma)_{Set}$ que cumplen que todas las álgebras de la familia poseen los mismos soportes para los géneros de Σ. Por tanto, es claro que el funtor no es sobreyectivo sobre objetos. Tampoco es pleno, ya que los morfismos que están en la imagen del funtor son morfismos (h, H) en los que la aplicación H es constante, propiedad que, en general, no posee cualquier morfismo de $\mathcal{C}(\Sigma)_{Set}$.

Veamos cuáles son los objetos de las categorías construidas a partir de las mostradas en el ejemplo 2.7.2.

Ejemplo 2.7.3

1. Consideramos la categoría $\mathcal{C}(\texttt{GRP})$, definida en el ejemplo 2.7.2, y cuyos objetos son los grupos. Los objetos de la categoría $\mathcal{C}(\texttt{GRP}_{imp})$, asociada a $\mathcal{C}(\texttt{GRP})$, son las $\texttt{GRP}_{imp}$-álgebras descritas del siguiente modo:

$$Obj(\mathcal{C}(\texttt{GRP}_{imp})) = \big\{\langle A, D, f\rangle \;\mid\; \forall a \in A, \langle D, f(a,-)\rangle \text{ es grupo}\big\}$$

   donde, $f = (f_{prd}\colon A \times D \times D \to D, f_{inv}\colon A \times D \to D, f_{unt}\colon A \to D)$ y, para todo $a \in A$, $f(a,-) = (f_{prd}(a,-)\colon D \times D \to D, f_{inv}(a,-)\colon D \to D, f_{unt}(a,-)\colon \{*\} \to D)$. Por tanto, cada $\texttt{GRP}_{imp}$-álgebra de $\mathcal{C}(\texttt{GRP}_{imp})$ determina una familia de grupos de forma que todos los de cada familia son grupos sobre el mismo conjunto, el soporte de la $\texttt{GRP}_{imp}$-álgebra para el género g.

2. A partir del TAD $\mathcal{T}_{\texttt{ANLL}}$ presentado en el ejemplo 2.7.2, consideramos la categoría $\mathcal{C}(\texttt{ANLL}_{imp})$ de $\texttt{ANLL}_{imp}$-álgebras verificando que, cada dato del soporte del género distinguido, que denotamos por $imp_{\texttt{ANLL}}$, define, de forma análoga a la descrita anteriormente para grupos, un anillo. Así, cada objeto de la categoría $\mathcal{C}(\texttt{ANLL}_{imp})$ representa a una familia de anillos, todos sobre el mismo conjunto base.

□

2.7.2 Subcategorías de álgebras con soportes fijos y objetos finales

Lo mismo que ocurre al trabajar con todo $Alg(\Sigma)$, el objeto final $\mathbb{1}_{\mathcal{C}(\Sigma)}\colon Obj(\mathcal{C}(\Sigma)) \to Obj(\mathcal{C}(\Sigma))$ no pertenece a la imagen de $I^{imp}_{\mathcal{C}(\Sigma_{imp})}$. No obstante, se puede hacer un desarrollo paralelo a lo estudiado para $Alg(\Sigma)$, considerando subcategorías con soportes fijos para los géneros de Σ.

Ya que se trata de una adaptación de lo realizado en la sección 2.4, nos vamos a limitar a presentar los resultados, procurando evitar notación y detalles que podrían resultar innecesarios y aburridos para el lector.

Si G es el conjunto de géneros de la signatura Σ, para cada G-conjunto D, nos restringimos a la subcategoría $\mathcal{C}^D(\Sigma)$ de $\mathcal{C}(\Sigma)$ formada por las álgebras cuyos soportes son los conjuntos D. Del mismo modo, tomamos la correspondiente subcategoría $\mathcal{C}^D(\Sigma_{imp})$ de $\mathcal{C}(\Sigma_{imp})$. Usando el funtor $I^{imp}_{\mathcal{C}(\Sigma_{imp})}$ se tiene:

Proposición 2.7.4 *Para cada G-conjunto D, el funtor $I^{imp}_{\mathcal{C}(\Sigma_{imp})}$ induce una biyección entre los objetos de las categorías $\mathcal{C}^D(\Sigma_{imp})$ y $\mathcal{C}^D(\Sigma)_{Set}$.*

Proposición 2.7.5 *Los únicos morfismos de $\mathcal{C}^D(\Sigma)_{Set}$ que son imagen por $I^{imp}_{\mathcal{C}(\Sigma_{imp})}$ de morfismos de $\mathcal{C}^D(\Sigma_{imp})$ son aquellos (h, H) tal que H es constante.*

La biyección a la que hace referencia la proposición 2.7.4 nos permite considerar como objeto distinguido en $\mathcal{C}^D(\Sigma_{imp})$ la imagen inversa de $\mathbb{1}_{\mathcal{C}^D(\Sigma)}$. Este objeto admite como descripción la siguiente:

$$\left\langle Obj(\mathcal{C}^D(\Sigma)), D, \{imp\_\sigma_{\mathbb{1}_{\mathcal{C}^D(\Sigma)}} : Obj(\mathcal{C}^D(\Sigma)) \times D_\omega \to D_v\}_{(\sigma:\, \omega\to v)\in\Sigma} \right\rangle$$

estando las operaciones definidas, para todo $A = \left\langle D, \{\sigma_A \colon D_\omega \to D_v\}_{(\sigma:\, \omega\to v)\in\Sigma} \right\rangle$ objeto de $\mathcal{C}^D(\Sigma)$ y para todo $d_\omega \in D_\omega$, como $imp\_\sigma_{\mathbb{1}_{\mathcal{C}^D(\Sigma)}}(A, d_\omega) := \sigma_A(d_\omega)$.

En términos de tuplas de funciones podemos interpretarlo del siguiente modo:

$$\mathbb{F} = \left\{ (f_\sigma \colon D_\omega \to D_v)_{(\sigma:\, \omega\to v)\in\Sigma} \in \prod_{(\sigma:\, \omega\to v)\in\Sigma} D_v^{D_\omega} \;\mid\; \langle D, \{f_\sigma\}_{\sigma\in\Sigma}\rangle \in \mathcal{C}^D(\Sigma) \right\}$$

por lo que, a partir del conjunto $\mathbb{F}$, se tiene otra descripción del objeto $\mathbb{1}_{\mathcal{C}^D(\Sigma)}$ como

$$\left\langle \mathbb{F}, D, \{imp\_\sigma_{\mathbb{1}_{\mathcal{C}^D(\Sigma)}} : F \times D_\omega \to D_v\}_{(\sigma:\, \omega\to v)\in\Sigma} \right\rangle$$

siendo $imp\_\sigma_{\mathbb{1}^D}\big((f_\delta)_{(\delta:\, \omega'\to v')\in\Sigma}, d_\omega\big) := f_\sigma(d_\omega)$, para todo $(f_\delta)_{(\delta:\, \omega'\to v')\in\Sigma} \in \mathbb{F}$ y para todo $d_\omega \in D_\omega$.

La existencia de la inclusión $I^{imp}_{\mathcal{C}(\Sigma_{imp})}$ junto con la aplicación del teorema 2.3.3 a las categorías anteriores, permite obtener la siguiente propiedad del objeto $\mathbb{1}_{\mathcal{C}^D(\Sigma)}$:

Teorema 2.7.6 *El objeto $\mathbb{1}_{\mathcal{C}^D(\Sigma)}$ es el coproducto en $\mathcal{C}^{D,\{\}}(\Sigma_{Imp})$ de los objetos de la imagen de la inclusión canónica de $\mathcal{C}^D(\Sigma)$ en $\mathcal{C}^D(\Sigma_{Imp})$.*

Este objeto no es final, siempre existe al menos un morfismo desde cualquier objeto de $\mathcal{C}^D(\Sigma)_{Set}$ en él, pero no necesariamente se tiene unicidad. Para tener unicidad, al igual que se hizo en la sección 2.4, hay que restringirse a tomar solo identidades sobre $\mathcal{C}^D(\Sigma)$ y aplicar luego las operaciones $()_{Set}$ e $()_{imp}$, para las que se obtienen sendas categorías isomorfas, a las que denotamos $\mathcal{C}^{D,\{\}}(\Sigma)_{Set}$ y $\mathcal{C}^{D,\{\}}(\Sigma_{imp})$ respectivamente. Con ello tenemos:

Teorema 2.7.7 *El objeto $\mathbb{1}_{\mathcal{C}^D(\Sigma)}$ es final en la categoría $\mathcal{C}^{D,\{\}}(\Sigma_{imp})$.*

Tomando $\mathcal{C}^{\{\}}(\Sigma_{imp})$, la subcategoría de $\mathcal{C}(\Sigma_{imp})$ cuyos morfismos son la identidad sobre los soportes correspondientes a los géneros de G, dejando libertad solamente para el género distinguido, se cumple:

Teorema 2.7.8 *La familia* $\{\mathbb{1}_{\mathcal{C}^D(\Sigma)}\}_{D\in\wp(\mathcal{U})^G}$ *es final en la categoría* $\mathcal{C}^{\{\}}(\Sigma_{imp})$.

Ejemplo 2.7.9

1. Continuando con el ejemplo 2.7.2, para cada D subconjunto de $\mathcal{U}$, denotamos por $\mathcal{C}^D(\texttt{GRP})$ a la subcategoría plena de $\mathcal{C}(\texttt{GRP})$ cuyas álgebras poseen a D como soporte para el género g. Dicho de otra forma, los objetos de $\mathcal{C}^D(\texttt{GRP})$ son los grupos sobre el conjunto D. Consideramos la categoría $\mathcal{C}^D(\texttt{GRP}_{imp})$ definida a partir de la categoría $\mathcal{C}^D(\texttt{GRP})$, tal y como se ha hecho en la sección 2.7.1. Una descripción del objeto final $\mathbb{1}_{\mathcal{C}^D(\texttt{GRP}_{imp})}$ de la categoría $\mathcal{C}^{D,\{\}}(\texttt{GRP}_{imp})$ es la siguiente:

   - El soporte para el género distinguido $imp_{\texttt{GRP}}$ es el conjunto de grupos sobre D, conjunto que denotamos por $\mathcal{G}^D$. Expresaremos cada elemento de $\mathcal{G}^D$ como $\langle D, f\rangle$, donde $f = (f_{prd}\colon D\times D\to D, f_{inv}\colon D\to D, f_{unt}\colon \{*\}\to D) \in D^{D\times D}\times D^D\times D$ verifica las propiedades de grupo. Así, la descripción del soporte para el género $imp_{\texttt{GRP}}$ es la siguiente:

$$\mathcal{G}^D = \big\{\langle D, f\rangle \;\mid\; \langle D, f\rangle \text{ es grupo}\big\}$$

   - Las operaciones vienen definidas por:
     · $imp\_prd_{\mathbb{1}_{\mathcal{C}^D(\texttt{GRP}_{imp})}} : \mathcal{G}^D\times D\times D\to D$, dada, para todo $\langle D, f\rangle\in\mathcal{G}^D$ y para todo $d_1, d_2\in D$, por $imp\_prd_{\mathbb{1}_{\mathcal{C}^D(\texttt{GRP}_{imp})}}\big(\langle D, f\rangle, d_1, d_2\big) := f_{prd}(d_1, d_2)$;
     · $imp\_inv_{\mathbb{1}_{\mathcal{C}^D(\texttt{GRP}_{imp})}} : \mathcal{G}^D\times D\to D$ definida por $imp\_inv_{\mathbb{1}_{\mathcal{C}^D(\texttt{GRP}_{imp})}}\big(\langle D, f\rangle, d_1\big) := f_{inv}(d_1)$, para todo $\langle D, f\rangle\in\mathcal{G}^D$ y para todo $d_1\in D$; y, finalmente,
     · $imp\_unt_{\mathbb{1}_{\mathcal{C}^D(\texttt{GRP}_{imp})}} : \mathcal{G}^D\to D$ definida, para todo $\langle D, f\rangle\in\mathcal{G}^D$ como $imp\_unt_{\mathbb{1}_{\mathcal{C}^D(\texttt{GRP}_{imp})}}\big(\langle D, f\rangle\big) := f_{unt}(*)$.

   Denotando por $\mathbb{F}^D_{\texttt{GRP}}$ al siguiente conjunto:

$$\mathbb{F}^D_{\texttt{GRP}} = \Big\{f = (f_{prd}\colon D\times D\to D, f_{inv}\colon D\to D, f_{unt}\colon\{*\}\to D)\in D^{D\times D}\times D^D\times D \;\mid$$

$$\mid\; \langle D, f\rangle \text{ es grupo de } \mathcal{C}^D(\texttt{GRP})\Big\}$$

   es evidente que $\mathbb{F}^D_{\texttt{GRP}}$ es biyectivo con el conjunto $\mathcal{G}^D$, lo que permite obtener otra descripción del objeto $\mathbb{1}_{\mathcal{C}^D(\texttt{GRP}_{imp})}$. La idea de esta descripción radica en el hecho de que, teniendo fijo el dominio D, un grupo queda caracterizado únicamente por sus operaciones.

2. En el caso del TAD $\mathcal{T}_{\texttt{ANLL}}$, cada conjunto D nos permite considerar la categoría $\mathcal{C}^D(\texttt{ANLL})$ cuyos objetos son los anillos sobre el conjunto D. La expresión del objeto final $\mathbb{1}_{\mathcal{C}^D(\texttt{ANLL}_{imp})}$ en la categoría $\mathcal{C}^{D,\{\}}(\texttt{ANLL}_{imp})$ es la siguiente:

- El soporte para el género distinguido $imp_{\mathtt{ANLL}}$ es el conjunto formado por los anillos sobre D, conjunto que denotamos por $\mathcal{A}^D$. Así,

$$\mathcal{A}^D = \big\{\langle D, f\rangle \ \mid \ \langle D, f\rangle \text{ es anillo}\big\}$$

  por tanto, $f = (f_{sum}\colon D \times D \to D, f_{opto}\colon D \to D, f_{ntr}\colon \{*\} \to D, f_{prd}\colon D \times D \to D, f_{unt}\colon \{*\} \to D) \in D^{D\times D} \times D^D \times D \times D^{D\times D} \times D$ verificando las propiedades de anillo.

- La definición de las operaciones es la siguiente:
  - La función $imp\_sum_{\mathbb{1}_{\mathcal{C}^D(\mathtt{ANLL}_{imp})}} : \mathcal{A}^D \times D \times D \to D$ se define por $imp\_sum_{\mathbb{1}_{\mathcal{C}^D(\mathtt{ANLL}_{imp})}}\big(\langle D, f\rangle, d_1, d_2\big) := f_{sum}(d_1, d_2)$;
  - $imp\_opto_{\mathbb{1}_{\mathcal{C}^D(\mathtt{ANLL}_{imp})}} : \mathcal{A}^D \times D \to D$ dada por $imp\_opto_{\mathbb{1}_{\mathcal{C}^D(\mathtt{ANLL}_{imp})}}\big(\langle D, f\rangle, d_1\big) := f_{opto}(d_1)$;
  - $imp\_ntr_{\mathbb{1}_{\mathcal{C}^D(\mathtt{ANLL}_{imp})}} : \mathcal{A}^D \to D$ definida como $imp\_ntr_{\mathbb{1}_{\mathcal{C}^D(\mathtt{ANLL}_{imp})}}\big(\langle D, f\rangle\big) := f_{ntr}(*)$;
  - La función $imp\_prd_{\mathbb{1}_{\mathcal{C}^D(\mathtt{ANLL}_{imp})}} : \mathcal{A}^D \times D \times D \to D$ viene dada por $imp\_prd_{\mathbb{1}_{\mathcal{C}^D(\mathtt{ANLL}_{imp})}}\big(\langle D, f\rangle, d_1, d_2\big) := f_{sum}(d_1, d_2)$; y, finalmente
  - $imp\_unt_{\mathbb{1}_{\mathcal{C}^D(\mathtt{ANLL}_{imp})}} : \mathcal{A}^D \to D$ definida como $imp\_unt_{\mathbb{1}_{\mathcal{C}^D(\mathtt{ANLL}_{imp})}}\big(\langle D, f\rangle\big) := f_{unt}(*)$;

  para todo $\langle D, f\rangle \in \mathcal{A}^D$ y para todo $d_1, d_2 \in D$.

Análogamente a lo visto en el ejemplo de los grupos, el conjunto

$$\mathbb{F}^D_{\mathtt{ANLL}} = \Big\{ f = (f_{sum}\colon D \times D \to D, f_{opto}\colon D \to D, f_{ntr}\colon \{*\} \to D, f_{prd}\colon D \times D \to D,$$

$$f_{unt}\colon \{*\} \to D) \in D^{D\times D} \times D^D \times D \times D^{D\times D} \times D \ \mid \ \langle D, f\rangle \text{ es anillo de } \mathcal{C}^D(\mathtt{ANLL})\Big\}$$

es biyectivo con el conjunto $\mathcal{A}^D$. Con $\mathbb{F}^D_{\mathtt{ANLL}}$ como soporte para $imp_{\mathtt{ANLL}}$, la definición de las operaciones es evidente.

□

2.7.3 Una operación sobre Tipos Abstractos de Datos

Lo tratado en el apartado anterior nos va a permitir extender a TADs la operación $()_{imp}$ definida, hasta ahora, por una parte sobre signaturas, y por otra sobre categorías de álgebras. Para ello definimos, a partir de un TAD $\mathcal{T}$, otro TAD que denotamos por $\mathcal{T}_{imp}$ con el que se pretende especificar las familias de álgebras que pueden obtenerse a partir del TAD $\mathcal{T}$. Como veremos, lo anterior corresponde en el paso a la implementación, a la idea de especificar las estructuras de datos correspondientes a las distintas formas de trabajar en la máquina con las representaciones de álgebras del TAD $\mathcal{T}$, es decir, a la especificación de implementaciones de $\mathcal{T}$.

A partir del TAD $\mathcal{T} = \langle \Sigma, \mathcal{C}(\Sigma)\rangle$ definimos el TAD $\mathcal{T}_{imp} = \langle \Sigma_{imp}, \mathcal{C}(\Sigma_{imp})\rangle$. Recordar que:

- Σ_{imp} es la signatura obtenida aplicando a la signatura Σ la operación $()_{imp}$;

- $\mathcal{C}(\Sigma_{imp})$ es la subcategoría de $Alg(\Sigma_{imp})$ construida a partir de la subcategoría $\mathcal{C}(\Sigma)$ de $Alg(\Sigma)$, tal y como se ha mostrado en el apartado 2.7.1.

El TAD $\mathcal{T}_{imp}$ modela a las familias de Σ-álgebras del TAD $\mathcal{T}$.

Ejemplo 2.7.10

1. Continuando con el ejemplo 2.7.2, a partir del TAD $\mathcal{T}_{\texttt{GRP}} = \langle \texttt{GRP}, \mathcal{C}(\texttt{GRP})\rangle$ que modela a los grupos, podemos definir el TAD $\mathcal{T}_{\texttt{GRP}_{imp}}$ descrito por el par $\mathcal{T}_{\texttt{GRP}_{imp}} = \langle \texttt{GRP}_{imp}, \mathcal{C}(\texttt{GRP}_{imp})\rangle$, donde $\mathcal{C}(\texttt{GRP}_{imp})$ está formada por aquellas $\texttt{GRP}_{imp}$-álgebras que especifican a familias de grupos, con todos los grupos de la familia sobre el mismo conjunto.

2. Análogamente, podemos definir a partir del TAD $\mathcal{T}_{\texttt{ANLL}}$ que modela a los anillos, el TAD $\mathcal{T}_{\texttt{ANLL}_{imp}}$ que especifica a las familias de anillos sobre un mismo conjunto.

□

Para cada G-conjunto D fijo, podemos definir, a partir del TAD $\mathcal{T} = \langle \Sigma, \mathcal{C}(\Sigma)\rangle$ otros TADs. Por una parte, el TAD $\mathcal{T}^D = \langle \Sigma, \mathcal{C}^D(\Sigma)\rangle$; por otra, el TAD $\mathcal{T}^D_{imp} = \langle \Sigma_{imp}, \mathcal{C}^D(\Sigma_{imp})\rangle$. El TAD $\mathcal{T}^D_{imp}$ modela a las familias de Σ-álgebras del TAD $\mathcal{T}$ que poseen a D como soporte para los géneros de Σ.

El TAD $\mathcal{T}_{imp}$ contiene al objeto $\mathbb{1}_{\mathcal{C}(\Sigma)}$, objeto que, en cierto sentido recoge a todos los del TAD $\mathcal{T}$. Para cada G-conjunto D, el objeto $\mathbb{1}_{\mathcal{C}^D(\Sigma)}$ está en el TAD $\mathcal{T}^D_{imp}$, objeto final en $\mathcal{C}^{D,\{\}}(\Sigma_{imp})$ y que "contiene" a todos los objetos del TAD $\mathcal{T}^D$. La propiedad de finalidad del objeto $\mathbb{1}_{\mathcal{C}^D(\Sigma)}$ hace que trabajar con él sea suficiente cuando lo que se necesite sea trabajar con todas las álgebras del TAD $\mathcal{T}^D$. Todo esto quedará más claro cuando lo interpretemos en el nivel de las implementaciones.

En los ejemplos anteriores, para cada conjunto D, el TAD $\mathcal{T}^D_{\texttt{GRP}}$ contiene a los grupos sobre el conjunto D, mientras que el TAD $\mathcal{T}^D_{\texttt{GRP}_{imp}} = \langle \texttt{GRP}_{imp}, \mathcal{C}^D(\texttt{GRP}_{imp})\rangle$ modela a las familias de grupos sobre el conjunto D.

2.8 El paso a álgebras parciales

Hasta ahora, dada una signatura Σ, hemos trabajado con álgebras de la categoría $Alg(\Sigma)$, álgebras cuyas operaciones son totales. Sin embargo, es habitual encontrar situaciones en las que los modelos a especificar contienen operaciones parciales. (El libro [22] es una referencia para el estudio de álgebras parciales.)

De hecho, en el marco de especificación algebrica hay trabajos que también tratan la parcialidad. Algunos de ellos son [90], [55], [24], [19], [5] y [21].

En el primer capítulo, introdujimos las nociones básicas de parcialidad y las utilizadas en especificación algebraica considerando parcialidad. Recordar que, dada una signatura $\Sigma = \langle G, \Omega\rangle$, una Σ-*álgebra parcial* consta de un conjunto asociado a cada género de Σ, denominado *soporte* para el género, y de una función, posiblemente parcial, para cada una de las operaciones de Σ, función definida entre los soportes correspondientes.

En el caso de la signatura Σ_{imp}, construida a partir de la signatura Σ, es conveniente trabajar con álgebras parciales. La razón fundamental es que la parcialidad aparece de manera natural

al fijar los soportes para los géneros de Σ. Esto es debido a que una vez fijado el G-conjunto D (formado por los soportes para los géneros de Σ), puede ocurrir que para cada elemento del soporte correspondiente al género imp_Σ, no interese considerar todos los elementos de los soportes de los otros géneros. Inicialmente, el hecho de fijar un G-conjunto D puede parecer artificial, sin embargo, como a continuación comentamos, no lo es ya que viene motivado por lo que habitualmente se hace en la práctica, en particular, es así como se trabaja en EAT. En dicho software, todos los elementos de las estructuras tienen un formato fijo, que es el que define el G-conjunto D, de ahí la necesidad de fijarlo. Pero, en general, no todos los elementos de D tendrán sentido para todas las estructuras consideradas, y es aquí donde aparece de manera natural la parcialidad.

Por ejemplo, volviendo a la signatura GRP introducida en el ejemplo 2.2.1, habiendo fijado $\mathbb{Z}$ como soporte para el género g, género correspondiente a los elementos de los grupos, suponer que se pretende dar una especificación de la familia de GRP-álgebras $\{\mathbb{Z}/n\mathbb{Z}\}_{n\in\mathbb{N},n>1}$. Definimos una $\mathtt{GRP}_{imp}$-álgebra parcial que llamamos A y que representa a la familia anterior. Para definir A, tomamos el conjunto $\mathbb{N}\setminus\{0,1\}$ como soporte para el género $imp_{\mathtt{GRP}}$. Para cada $n\in\mathbb{N}$, definimos el conjunto $< n >= \{0,\ldots,n-1\} \subsetneqq D_g$, conjunto que utilizaremos para dar los dominios de definición de las operaciones del álgebra A. Consideramos la $\mathtt{GRP}_{imp}$-álgebra parcial $A = \langle \mathbb{N}\setminus\{0,1\}, \mathbb{Z}, \{imp\_prd_A, imp\_inv_A, imp\_unt_A\}\rangle$ siendo las operaciones:

- $imp\_prd_A(n, z1, z2) = (z1 + z2) \bmod n$, operación parcial con dominio $\{(n, z1, z2) \in \mathbb{N}\setminus\{0,1\} \times \mathbb{Z} \times \mathbb{Z} \mid z_1, z_2 \in < n > \}$;
- $imp\_inv_A(n, z) = -z \bmod n$, con $Def(imp\_inv_A) = \{(n, z) \in \mathbb{N}\setminus\{0,1\}\times\mathbb{Z} \mid z \in < n > \}$;
- $imp\_unt_A(n) = 0$, con dominio $\mathbb{N}\setminus\{0,1\}$.

Es claro que A representa a la familia de grupos $\{\mathbb{Z}/n\mathbb{Z}\}_{n\in\mathbb{N},n>1}$.

Sin embargo, esta álgebra no formará parte de ninguna de las categorías que vamos a estudiar. Como veremos más adelante, en el caso de álgebras parciales, las categorías en las que se obtienen los resultados son categorías formadas por álgebras parciales en las que los dominios de definición de las operaciones son cubos de la forma $S \times S_\omega$ con $S \subseteq A_{imp_\Sigma}$, siendo éste último el soporte correspondiente al género $imp_{\mathtt{GRP}}$ y $S_\omega \subseteq A_\omega$, con $\sigma\colon \omega \to v$. Por tanto, el álgebra A definida anteriormente no pertenecerá a ninguna de las categorías que consideraremos. Sin embargo, para cada $n \in \mathbb{N}\setminus\{0,1\}$, podemos obtener una $\mathtt{GRP}_{imp}$-álgebra parcial, que pertenecerá a alguna de las categorías que vamos a considerar, y que especifica una familia de GRP-álgebras (en este caso formada por una única álgebra).

Otro ejemplo podría ser el siguiente: como en el caso anterior, consideramos $\mathbb{Z}$ como soporte fijo para el género g, tomamos $\mathbb{N}\setminus\{0,1\}$ como soporte para el género $imp_{\mathtt{GRP}}$ y definimos la $\mathtt{GRP}_{imp}$-álgebra parcial $B = \langle \mathbb{N}\setminus\{0,1\}, \mathbb{Z}, \{imp\_prd_B, imp\_inv_B, imp\_unt_B\}\rangle$ con operaciones:

- $imp\_prd_B(n, z1, z2) = (z1 + z2) \bmod n$, para todo $n \in \mathbb{N}\setminus\{0,1\}$ y para todo $z_1, z_2 \in \mathbb{N}$;
- $imp\_inv_B(n, z) = -z \bmod n$, para todo $n \in \mathbb{N}\setminus\{0,1\}$ y para todo $z \in \mathbb{N}$; e
- $imp\_unt_B(n) = 0$, para todo $n \in \mathbb{N}\setminus\{0,1\}$.

La $\mathtt{GRP}_{imp}$-álgebra B representa a la misma familia de GRP-álgebras, $\{\mathbb{Z}/n\mathbb{Z}\}_{n\in\mathbb{N},n>1}$. Observar que todas las operaciones de la $\mathtt{GRP}_{imp}$-álgebra anterior son parciales y su dominio

de definición forma un cubo: $Def(imp\_prd_B) = \mathbb{N} \setminus \{0,1\} \times \mathbb{N} \times \mathbb{N} \subset \mathbb{N} \setminus \{0,1\} \times \mathbb{Z} \times \mathbb{Z}$, $Def(imp\_inv_B) = \mathbb{N} \setminus \{0,1\} \times \mathbb{N} \subset \mathbb{N} \setminus \{0,1\} \times \mathbb{Z}$ y $Def(imp\_unt_B) = \mathbb{N} \setminus \{0,1\}$.

En general, la idea es que el dominio de definición de cada operación $imp\_\sigma \colon A_{imp_{\mathtt{GRP}}} \times A_\omega \to A_v$ de una Σ_{imp}-álgebra, sea un conjunto de la forma $S \times S_\omega$ con $S \subseteq A_{imp_\Sigma}$ y $S_\omega \subseteq D_\omega$. Si con una Σ_{imp}-álgebra se pretende especificar una familia de álgebras totales, entonces $S_\omega = D_\omega$ y, en el caso particular de que $S = A_{imp_{\mathtt{GRP}}}$, se tiene una Σ_{imp}-álgebra total.

Vamos a estudiar a continuación el caso de especificación de familias de álgebras permitiendo parcialidad en las operaciones. En el paso a implementaciones veremos que esto último es fundamental por dos motivos. Por una parte, para especificar implementaciones es necesario recurrir a álgebras parciales; por otra, en ocasiones está fijo el soporte para el género distinguido (caso habitual en implementaciones) y al permitir parcialidad en las operaciones pueden especificarse determinadas familias de álgebras que no podrían especificarse de otro modo. Vamos en esta sección a estudiar la categoría de Σ_{imp}-álgebras parciales cuyos objetos son familias de Σ-álgebras, que pueden ser totales o parciales. Para ello, generalizaremos lo visto en las secciones anteriores permitiendo parcialidad en las operaciones.

Si $\Sigma = \langle G, \Omega \rangle$ es una signatura y A es una Σ-álgebra parcial, la representaremos como:

$$A = \left\langle (A_g)_{g \in G}, \{\sigma_A \colon A_\omega \to A_v, Def(\sigma_A)\}_{(\sigma \colon \omega \to v) \in \Sigma} \right\rangle$$

donde, para cada $\sigma \in \Sigma$, $Def(\sigma_A) \subseteq A_\omega$ es el *dominio de definición de la función* σ_A.

Tal y como hemos comentado en los preliminares, existen diferentes nociones de morfismo entre álgebras parciales (ver por ejemplo [72]). La que aquí vamos a adoptar es la que habitualmente en la literatura, y concretamente en [72], se denomina morfismo débil (aunque nosotros lo denominaremos simplemente morfismo), noción que ya incluimos en los preliminares (definición 1.2.43) y que a continuación recordamos. Si $\Sigma = \langle G, \Omega \rangle$ es una signatura y A y B son Σ-álgebras parciales, un Σ-*morfismo* $f \colon A \to B$ *entre* A *y* B es una familia de funciones *totales* $f = (f_g)_{g \in G}$, con $f_g \colon A_g \to B_g$ para cada $g \in G$, verificando que, para cada $(\sigma \colon \omega \to v) \in \Sigma$ y cada $a \in A_\omega$, si $\sigma_A(a)$ está definido entonces $\sigma_B(f_\omega(a))$ también lo está y, además, $f_v(\sigma_A(a)) = \sigma_B(f_\omega(a))$.

Denotamos por $PAlg(\Sigma)$ a la categoría de Σ-álgebras parciales junto con los morfismos entre ellas. Observar que $Alg(\Sigma)$ es una subcategoría plena de $PAlg(\Sigma)$. Por tanto, podíamos haber comenzado el estudio permitiendo parcialidad, sin embargo hemos preferido hacerlo en dos pasos para no agrupar todas las dificultades en la exposición.

El objetivo de esta sección es estudiar propiedades de la categoría $PAlg(\Sigma_{imp})$ con el fin de determinar la existencia de objetos que especifiquen familias de álgebras (parciales) lo más amplias y generales posible. Por ello, a partir de ahora trabajaremos en el marco parcial. En el resto de la sección, al referirnos a álgebras estaremos considerando álgebras parciales y, por tanto, al hablar de morfismos, serán morfismos entre álgebras parciales. También consideraremos que los soportes de las álgebras son elegidos en $\mathcal{U}$, el universo algebraico.

2.8.1 Familias de Σ-álgebras representadas por Σ_{imp}-álgebras en el caso parcial. Resultados de finalidad

Partimos de una signatura $\Sigma = \langle G, \Omega \rangle$ y de la categoría de álgebras $PAlg(\Sigma)$. A partir de esta categoría podemos considerar otras dos. Por un lado, la categoría de Σ_{imp}-álgebras $PAlg(\Sigma_{imp})$, categoría objeto de nuestro estudio, y, por otro, la categoría $PAlg(\Sigma)_{Set}$, que nos permitirá deducir propiedades de la anterior. Notar que aunque ahora estemos trabajando en el marco

de funciones parciales, la operación $()_{Set}$ es la misma que definimos en el apartado 2.3.1, por lo que, los objetos de la categoría $PAlg(\Sigma)_{Set}$ son funciones totales.

Una generalización natural del funtor I^{imp} nos permite relacionar ambas categorías. Consideramos $I^{imp}\colon PAlg(\Sigma_{imp}) \to PAlg(\Sigma)_{Set}$ que a cada Σ_{imp}-álgebra parcial $A = \langle A_{imp_\Sigma}, (A_g)_{g\in G}, \{imp\_\sigma_A\colon A_{imp_\Sigma} \times A_\omega \to A_v, Def(imp\_\sigma_A)\}_{(\sigma\colon \omega\to v)\in\Sigma}\rangle$ le asocia el correspondiente objeto $I^{imp}(A)$ dado por la aplicación $\alpha_{A_{imp_\Sigma}}\colon A_{imp_\Sigma} \to Obj(PAlg(\Sigma))$, definida, para cada $a \in A_{imp_\Sigma}$, por $\alpha_{A_{imp_\Sigma}}(a) := \langle (A_g)_{g\in G}, \{imp\_\sigma_A(a,-)\colon A_\omega \to A_v, Def(imp\_\sigma_A(a,-))\}_{(\sigma\colon \omega\to v)\in\Sigma}\rangle$, siendo $Def(imp\_\sigma_A(a,-)) = \{a_\omega \in A_\omega \mid (a, a_\omega) \in Def(imp\_\sigma_A)\}$. Sobre los morfismos, el funtor se define de la misma forma que en el caso total. Observar que si $a \notin Def(imp\_\sigma_A)$, entonces $Def(imp\_\sigma_A(a,-)) = \emptyset$.

El resultado recogido en el siguiente lema es el que garantiza que I^{imp} es un funtor.

Lema 2.8.1 *Sean* $A = \langle A_{imp_\Sigma}, (A_g)_{g\in G}, \{imp\_\sigma_A\colon A_{imp_\Sigma} \times A_\omega \to A_v, Def(imp\_\sigma_A)\}_{(\sigma\colon \omega\to v)\in\Sigma}\rangle$ *y* $B = \langle B_{imp_\Sigma}, (B_g)_{g\in G}, \{imp\_\sigma_B\colon B_{imp_\Sigma} \times B_\omega \to B_v, Def(imp\_\sigma_B)\}_{(\sigma\colon \omega\to v)\in\Sigma}\rangle$ *Σ_{imp}-álgebras, y sea* $f\colon A \to B$, $f = (f_{imp_\Sigma}, (f_g)_{g\in G})$, *un Σ_{imp}-morfismo. Entonces, para cada* $a \in A_{imp_\Sigma}$, $(f_g)_{g\in G}$ *es un morfismo en* $PAlg(\Sigma)$ *entre* $\alpha_{A_{imp_\Sigma}}(a)$ *y* $\alpha_{B_{imp_\Sigma}}(f_{imp_\Sigma}(a))$.

Cada Σ_{imp}-álgebra representa a una familia de Σ-álgebras, todas ellas con los mismos soportes para los géneros de Σ. Esto último ilustra el hecho de que el funtor I^{imp} no es sobreyectivo sobre objetos y da una idea de cómo agrupar los objetos dentro de la categoría $PAlg(\Sigma_{imp})$. De modo análogo al caso total, vamos a descomponer la categoría $PAlg(\Sigma_{imp})$ en subcategorías cuyas álgebras tengan los mismos soportes para los géneros de Σ. Ya que salvo pequeñas comprobaciones relacionadas con la parcialidad, todo resulta similar al caso total, procuraremos evitar al lector detalles innecesarios.

Para cada G-conjunto D, consideramos las correspondientes categorías $PAlg^D(\Sigma)$, $PAlg^D(\Sigma_{imp})$ y $PAlg^D(\Sigma)_{Set}$. El funtor I^{imp} permite obtener el resultado recogido en la siguiente proposición, que relaciona las categorías $PAlg^D(\Sigma_{imp})$ y $PAlg^D(\Sigma)_{Set}$.

Proposición 2.8.2 *Para cada G-conjunto D, existe una biyección entre los conjuntos* $Obj(PAlg^D(\Sigma_{imp}))$ *y* $Obj(PAlg^D(\Sigma)_{Set})$.

Al igual que en el caso total, consideramos en $PAlg^D(\Sigma)_{Set}$ el objeto $\mathbb{1}^D\colon Obj(PAlg^D(\Sigma)) \to Obj(PAlg^D(\Sigma))$. La biyección recogida en la proposición anterior nos conduce a tomar como objeto distinguido en $PAlg^D(\Sigma_{imp})$ la antiimagen del objeto $\mathbb{1}^D$. Este objeto puede describirse del siguiente modo:

$$\left\langle Obj(PAlg^D(\Sigma)), D, \{imp\_\sigma_{\mathbb{1}^D}\colon Obj(PAlg^D(\Sigma)) \times D_\omega \to D_v, Def(imp\_\sigma_{\mathbb{1}^D})\}_{(\sigma\colon \omega\to v)\in\Sigma}\right\rangle$$

con $Def(imp\_\sigma_{\mathbb{1}^D}) = \{(A, d_\omega) \mid d_\omega \in Def(\sigma_A)\}$ y estando las operaciones definidas, para cada $A = \langle D, \{\sigma_A\colon D_\omega \to D_v\}_{(\sigma\colon \omega\to v)\in\Sigma}\rangle$ y para cada $d_\omega \in D_\omega$ con $(A, d_\omega) \in Def(imp\_\sigma_{\mathbb{1}^D})$, por $imp\_\sigma_{\mathbb{1}^D}(A, d_\omega) := \sigma_A(d_\omega)$. Abusando en la notación, si no hay lugar a confusión, nos referiremos también a este objeto por $\mathbb{1}^D$. (Se puede dar una expresión de este objeto, similar a la dada en la página 79, en términos de tuplas funcionales.)

Respecto de los morfismos, es claro que, en general, no existe una biyección entre los morfismos de ambas categorías. Lo que se tiene es una caracterización de los morfismos de

$PAlg^D(\Sigma)_{Set}$ que están en la imagen por el funtor I^{imp}, y esto, permite dar una descripción explícita de la categoría $PAlg^D(\Sigma_{imp})$.

Proposición 2.8.3 *$PAlg^D(\Sigma_{imp})$ es isomorfa a la subcategoría de $PAlg^D(\Sigma)_{Set}$ cuyos morfismos son los (h, H) de $PAlg^D(\Sigma)_{Set}$ con H aplicación constante.*

La proposición anterior caracteriza las familias de Σ-álgebras que pueden representarse como Σ_{imp}-álgebras.

De nuevo, el objeto $\mathbb{1}^D$ no es final en la categoría $PAlg^D(\Sigma)_{Set}$. Siempre existe al menos un morfismo desde cualquier otro objeto en $\mathbb{1}^D$, pero no es posible asegurar la unicidad. Lo mismo que en el marco total, tomamos las correspondientes categorías de álgebras parciales $PAlg^{D,\{\}}(\Sigma)$, $PAlg^{D,\{\}}(\Sigma_{imp})$ y $PAlg^{D,\{\}}(\Sigma)_{Set}$, cuyas definiciones se describen en la tabla 2.3.

	Categoría	**Fijado D como G-conjunto**	**Morfismos que sobre los soportes de Σ son la identidad**	**Fijado D como G-conjunto y con morfismos identidad sobre D**
Operación	$PAlg(\Sigma)$	$PAlg^D(\Sigma)$	$PAlg^{\{\}}(\Sigma)$	$PAlg^{D,\{\}}(\Sigma)$
$-_{Set}$	$PAlg(\Sigma)_{Set}$	$PAlg^D(\Sigma)_{Set}$	$PAlg^{\{\}}(\Sigma)_{Set}$	$PAlg^{D,\{\}}(\Sigma)_{Set}$
$()_{imp}$	$PAlg(\Sigma_{imp})$	$PAlg^D(\Sigma_{imp})$	$PAlg^{\{\}}(\Sigma_{imp})$	$PAlg^{D,\{\}}(\Sigma_{imp})$

Tabla 2.3: Categorías de álgebras parciales introducidas

Teniendo en cuenta la inclusión de $PAlg^D(\Sigma)$ como subcategoría de $PAlg^D(\Sigma_{imp})$ y el isomorfismo entre las categorías $PAlg^{D,\{\}}(\Sigma)_{Set}$ y $PAlg^{D,\{\}}(\Sigma_{imp})$, deducido a partir de la proposición 2.8.3, la aplicación del teorema 2.3.3 lleva al siguiente resultado:

Teorema 2.8.4 *El objeto $\mathbb{1}^D$ es el coproducto en $PAlg^{D,\{\}}(\Sigma_{imp})$ de los objetos de la imagen de la inclusión canónica de $PAlg^D(\Sigma)$ en $PAlg^D(\Sigma_{imp})$.*

Respecto de la existencia de objeto final, para cada G-conjunto D, se tiene lo siguiente:

Teorema 2.8.5 *El objeto $\mathbb{1}^D$ es final en la categoría $PAlg^{D,\{\}}(\Sigma_{imp})$.*

Teniendo en cuenta que $Obj(PAlg^{\{\}}(\Sigma_{imp})) = \bigsqcup_{D \in \wp(\mathcal{U})^G} Obj(PAlg^{D,\{\}}(\Sigma_{imp}))$, los objetos $\mathbb{1}^D$ pueden agruparse por la siguiente propiedad universal.

Teorema 2.8.6 *La familia $\{\mathbb{1}^D\}_{D \in \wp(\mathcal{U})^G}$ es final en la categoría $PAlg^{\{\}}(\Sigma_{imp})$.*

2.8.2 Subcategorías de álgebras con dominios de definición de las operaciones fijos

Hasta este momento venimos trabajando a nivel de álgebras y hemos visto cómo el objeto $\mathbb{1}^D$ representaba, de algún modo, la familia más general de álgebras. Sin embargo, nuestro verdadero interés está, no tanto en el estudio de álgebras, sino en el de implementaciones de álgebras, a lo que dedicaremos el siguiente capítulo. En este punto señalamos una diferencia importante entre el trabajo con álgebras y el trabajo con implementaciones de álgebras.

Al expresar $\mathbb{1}^D$ no hemos tenido problema al establecer el dominio de las operaciones

$$Def(imp\_\sigma_{\mathbb{1}^D}) = \bigsqcup_{A \in Obj(PAlg^D(\Sigma))} \big[\{*\} \times Def(\sigma_A)\big]$$

Si pensamos en la descripción de $\mathbb{1}^D$ en la que el soporte para imp_Σ es un conjunto de tuplas funcionales, tampoco hay problema al expresar

$$Def(imp\_\sigma_{\mathbb{1}^D}) = \bigsqcup_{(f_\delta)_{\delta\in\Sigma} \in \prod_{(\sigma\colon \omega'\to v')\in\Sigma} D_{v'}^{D^{\omega'}}} \big[\{*\} \times Def(f_\sigma)\big]$$

Ahora bien, si pensamos en implementaciones y realizamos un desarrollo paralelo al realizado en el caso de las álgebras, la expresión del objeto final en una categoría adecuada de implementaciones nos llevaría a tener un dominio D ligado a un tipo de datos, y cada uno de los datos del soporte correspondiente a imp_Σ en el objeto final sería una tupla de códigos sintácticamente correctos sobre D. Por tanto, al pasar a implementaciones solo se tiene un código (que implementa la operación) y un dominio sobre el que el código es sintácticamente correcto, pero no se tiene el dominio real de definición, como ocurre al trabajar con funciones. Realmente, un mismo código puede tener distintos dominios de definición, por lo que en este caso se hace necesario fijar uno de ellos. Por ello, al trabajar con implementaciones llegados a este punto deberemos fijar dominios de definición para los códigos, que permitan determinar funciones. Esto no es necesario en el caso de trabajar con álgebras pero, por hacer explícito el paralelismo con las implementaciones, fijamos dominios de definición para las operaciones de las Σ-álgebras.

Fijamos un Ω-conjunto S, $S = (S^\sigma)_{(\sigma\colon \omega\to v)\in\Sigma}$ estando, para todo $(\sigma\colon \omega \to v) \in \Sigma$, $S^\sigma \subseteq D_\omega$.

Nos restringimos ahora a la subcategoría plena de $PAlg^{D,\{\}}(\Sigma)$ con objetos las Σ-álgebras que tienen como dominio de definición, para cada operación $\sigma \in \Sigma$, al conjunto S^σ, categoría que denotamos por $PAlg^{D,S,\{\}}(\Sigma)$. Análogamente, denotamos por $PAlg^{D,S,\{\}}(\Sigma_{imp})$ a la subcategoría plena de $PAlg^{D,\{\}}(\Sigma_{imp})$ con objetos las Σ_{imp}-álgebras que tienen, para cada operación imp_σ, al conjunto $S_{imp_\Sigma} \times S^\sigma$ como dominio de definición, siendo S_{imp_Σ} cualquier subconjunto del soporte para el género imp_Σ. Notar que hemos impuesto que los dominios de definición de las operaciones sean rectángulos con factor en la componente imp_Σ.

En la categoría $PAlg^{D,S,\{\}}(\Sigma_{imp})$ existe un objeto distinguido que denotamos por $\mathbb{1}^{D,S}$ y cuya descripción es la siguiente

$$\Big\langle Obj(PAlg^{D,S}(\Sigma)), D, \{imp\_\sigma_{\mathbb{1}^{D,S}}\colon Obj(PAlg^{D,S}(\Sigma)) \times D_\omega \to D_v, S_{imp_\Sigma} \times S^\sigma\}_{(\sigma\colon \omega\to v)\in\Sigma}\Big\rangle$$

Pues bien, resulta que el objeto $\mathbb{1}^{D,S}$ es final en la categoría $PAlg^{D,S,\{\}}(\Sigma_{imp})$, propiedad que recoge el siguiente teorema.

Teorema 2.8.7 *Para cada Ω-conjunto $S \subseteq (D_\omega)_{(\sigma\colon \omega\to v)\in\Sigma}$, el objeto $\mathbb{1}^{D,S}$ es final en $PAlg^{D,S,\{\}}(\Sigma)$.*

Vamos a estudiar cómo se relacionan estos objetos entre sí y con el objeto $\mathbb{1}^D$, objeto distinguido en $PAlg^D(\Sigma_{imp})$. La relación vendrá expresada en forma de propiedad universal de coproducto. Para ello, observar, en primer lugar, que la categoría $PAlg^{D,\{\}}(\Sigma_{imp})$ posee coproductos que vienen inducidos por la existencia de coproductos en Set.

Sea J un conjunto y sea $\left\{A^j = \left\langle A^j_{imp_\Sigma}, D,\ \{imp\_\sigma_{A^j}\colon A^j_{imp_\Sigma} \times D_\omega \to D_v,\ Def(imp\_\sigma_{A^j})\}_{(\sigma\colon \omega\to v)\in\Sigma}\right\rangle\right\}_{j\in J}$ una familia de Σ_{imp}-álgebras. A partir de la familia anterior consideramos la siguiente Σ_{imp}-álgebra:

$$A \equiv \left[A^j\right]_{j\in J} = \left\langle \bigsqcup_{j\in J} A^j_{imp_\Sigma}, D, \{imp\_\sigma_A\colon \bigsqcup_{j\in J} A^j_{imp_\Sigma} \times D_\omega \to D_v, Def(imp\_\sigma_A)\}_{(\sigma\colon \omega\to v)\in\Sigma}\right\rangle$$

siendo $Def(imp\_\sigma_A) = \left\{(a, d_\omega) \mid (a, d_\omega) \in Def(imp\_\sigma_{A^k}) \text{ con } k \in J \text{ y tal que } a \in A^k\right\}$, y definida por $imp\_\sigma_A(a, d_\omega) := imp\_\sigma_{A^k}(a, d_\omega)$, para todo $(a, d_\omega) \in Def(imp\_\sigma_A)$.

Para cada $j \in J$, consideramos el morfismo inclusión $i^j\colon A^j \to A$, es decir, $i^j = (i^j_{imp_\Sigma}, id_D)$, con $i^j_{imp_\Sigma}\colon A^j_{imp_\Sigma} \to \bigsqcup_{j\in J} A^j_{imp_\Sigma}$ la inclusión canónica.

Entonces, el objeto A junto con la familia de morfismos $\{i^j\}_{j\in J}$ es el coproducto en $PAlg^{D,\{\}}(\Sigma_{imp})$ de la familia $\{A^j\}_{j\in J}$.

Observar que

$$\bigsqcup_{S\subseteq(D_\omega)_{(\sigma\colon \omega\to v)\in\Sigma}} Obj(PAlg^{D,S,\{\}}(\Sigma)) \subsetneq Obj(PAlg^{D,\{\}}(\Sigma))$$

ya que no todos los dominios de las operaciones de las Σ_{imp}-álgebras son rectángulos que factorizan por el género distinguido. Llamamos $PAlg^{D,\{\}}(\Sigma_{imp})^*$ a la subcategoría plena de $PAlg^{D,\{\}}(\Sigma_{imp})$ con objetos las Σ_{imp}-álgebras cuyos dominios de definición de las funciones son rectángulos. Así, para cada $(\sigma\colon \omega \to v) \in \Sigma$, el dominio de definición de imp_σ es $S_{imp_\Sigma} \times S^\sigma \subseteq D_{imp_\Sigma} \times D_\omega$, siendo D_{imp_Σ} el soporte del género distinguido. Observar que, para cualquier Ω-conjunto $S \subseteq (D_\omega)_{(\sigma\colon \omega\to v)\in\Sigma}$, $PAlg^{D,S,\{\}}(\Sigma_{imp})$ es subcategoría de $PAlg^{D,\{\}}(\Sigma_{imp})^*$.

La categoría $PAlg^{D,\{\}}(\Sigma_{imp})^*$ posee coproductos y el coproducto en ella coincide con el coproducto en $PAlg^{D,\{\}}(\Sigma_{imp})$, lo que permite *agrupar* todos los objetos distinguidos de las categorías $PAlg^{D,S,\{\}}(\Sigma_{imp})$.

Denotamos por $\mathbb{1}^{D,*}$ al objeto distinguido de la categoría $PAlg^{D,\{\}}(\Sigma_{imp})^*$ y, consideramos en esa categoría los objetos finales de las categorías $PAlg^{D,S,\{\}}(\Sigma_{imp})$ a las que se refiere el teorema 2.8.7. Con ello se tiene el siguiente resultado

Teorema 2.8.8 *El objeto $\mathbb{1}^{D,*}$ es el coproducto en $PAlg^{D,\{\}}(\Sigma_{imp})$ de la familia de objetos $\left\{\mathbb{1}^{D,S}\right\}_{S\subseteq(D_\omega)_{(\sigma\colon \omega\to v)\in\Sigma}}$.*

2.8.3 Tipos Abstractos de Datos

En la sección 2.7 extendimos la operación $()_{imp}$ a tipos abstractos de datos en el marco total. Finalizamos este capítulo con un breve comentario sobre la generalización de lo hecho en la sección 2.7 para el caso de trabajar con álgebras parciales.

Definición 2.8.9 *Un* Tipo Abstracto de Datos (TAD) *es un par* $\mathcal{T} = \langle \Sigma, \mathcal{C}(\Sigma) \rangle$ *formado por una signatura* Σ *y una subcategoría* $\mathcal{C}(\Sigma)$ *de* $PAlg(\Sigma)$*, cerrada por isomorfismos.*

Tal y como hemos comentado en el marco total, en ocasiones la subcategoría $\mathcal{C}(\Sigma)$ se tomará cerrada por isomorfismos en una subcategoría propia de $PAlg(\Sigma)$. De modo totalmente análogo a lo hecho en el caso total, se puede asignar a cada subcategoría $\mathcal{C}(\Sigma)$ de $PAlg(\Sigma)$, la correspondiente subcategoría $\mathcal{C}(\Sigma_{imp})$ de $PAlg(\Sigma_{imp})$, del siguiente modo:

- Los objetos son las Σ_{imp}-álgebras $A = \big\langle A_{imp_\Sigma}, (A_g)_{g\in G}, \{imp\_\sigma_A \colon A_{imp_\Sigma} \times A_\omega \to A_v, Def(imp\_\sigma_A)\}_{(\sigma\colon \omega\to v)\in\Sigma} \big\rangle$ tales que, para todo $a \in A_{imp_\Sigma}$, la Σ-álgebra (parcial) $A_a = \big\langle (A_g)_{g\in G}, \{imp\_\sigma_A(a,-) \colon A_\omega \to A_v, Def(imp\_\sigma_A(a,-))\}_{(\sigma\colon \omega\to v)\in\Sigma} \big\rangle$ con $Def(imp\_\sigma_A(a,-)) = \big\{a_\omega \mid (a,a_\omega) \in Def(imp\_\sigma_A)\big\}$, está en $\mathcal{C}(\Sigma)$. (Observar que si no existe ningún $a_\omega \in A_\omega$ tal que $(a,a_\omega) \in Def(imp\_\sigma_A)$, entonces $Def(imp\_\sigma_A(a,-)) = \emptyset$).
- Si A y B son Σ_{imp}-álgebras de $\mathcal{C}(\Sigma)$, un Σ_{imp}-morfismo $f\colon A \to B$, $f = (f_{imp_\Sigma}, (f_g)_{g\in G})$, es un morfismo de $\mathcal{C}(\Sigma_{imp})$ si, para todo $a \in A_{imp_\Sigma}$, $(f_g)_{g\in G}\colon A_a \to B_{f_{imp_\Sigma}(a)}$ es un morfismo de $\mathcal{C}(\Sigma)$.

Al fijar un G-conjunto D se cuenta con las correspondientes categorías de álgebras parciales con soportes D, a las que, abusando de notación, nos referiremos por $\mathcal{C}^D(\Sigma)$, $\mathcal{C}^D(\Sigma_{imp})$ y $\mathcal{C}^D(\Sigma)_{Set}$. Lo mismo ocurre con la generalización del funtor $I^{imp}_{\mathcal{C}(\Sigma_{imp})}$ y del objeto $\mathbb{1}_{\mathcal{C}(\Sigma)}$. Con un desarrollo similar al realizado en la sección 2.7, llegamos a la generalización de los resultados allí presentados.

Teorema 2.8.10 *El objeto* $\mathbb{1}_{\mathcal{C}^D(\Sigma)}$ *es el coproducto en* $\mathcal{C}^{D,\{\}}(\Sigma_{imp})$ *de los objetos de la imagen de la inclusión canónica de* $\mathcal{C}^D(\Sigma)$ *en* $\mathcal{C}^D(\Sigma_{imp})$*.*

Teorema 2.8.11 *El objeto* $\mathbb{1}_{\mathcal{C}^D(\Sigma)}$ *es final en la categoría* $\mathcal{C}^{D,\{\}}(\Sigma_{imp})$*.*

Si $\mathcal{C}^{\{\}}(\Sigma_{imp})$ denota a la subcategoría de $\mathcal{C}(\Sigma_{imp})$ con únicos morfismos aquellos que son las identidades sobre los soportes correspondientes a los géneros de la signatura Σ, el resultado recogido en el último teorema unido al hecho de que $Obj(\mathcal{C}^{\{\}}(\Sigma_{imp})) = \bigsqcup_{D\in\wp(\mathcal{U})^G} Obj(\mathcal{C}^{D,\{\}}(\Sigma_{imp}))$, permiten obtener una caracterización de los objetos $\mathbb{1}_{\mathcal{C}^D(\Sigma)}$ por una propiedad universal que explica cómo se *agrupan* todos esos objetos finales.

Teorema 2.8.12 *La familia* $\{\mathbb{1}_{\mathcal{C}^D(\Sigma)}\}_{D\in\wp(\mathcal{U})^G}$ *es final en la categoría* $\mathcal{C}^{\{\}}(\Sigma_{imp})$*.*

Además, esto nos permite extender la operación $()_{imp}$ a TADs de álgebras parciales.

Sea $\mathcal{T} = \langle \Sigma, \mathcal{C}(\Sigma) \rangle$ un TAD. A partir de él definimos el TAD $\mathcal{T}_{imp}$ como el TAD siguiente: Σ_{imp} es la signatura obtenida aplicando a la signatura Σ la operación $()_{imp}$; y $\mathcal{C}(\Sigma_{imp})$ es la subcategoría de $PAlg(\Sigma_{imp})$ construida, tal y como se ha hecho al comienzo de la sección 2.8.3, a partir de la subcategoría $\mathcal{C}(\Sigma)$ de $PAlg(\Sigma)$.

El TAD $\mathcal{T}_{imp}$ modela a las familias de Σ-álgebras (parciales) del TAD $\mathcal{T}$.

2.9 Aplicaciones al Cálculo Simbólico

A continuación desarrollamos tres ejemplos más completos y próximos a lo que podemos encontrar en el sistema EAT. En estos ejemplos lo que se pretende es modelar categorías bien definidas

en Matemáticas, de forma que posteriormente se puedan utilizar las descripciones obtenidas para trabajar con ellas en la máquina. Intentamos ilustrar la dificultad existente al trabajar en la máquina con algunas estructuras que matemáticamente están bien definidas y cuyo uso no plantea ningún problema al matemático. El primero de los ejemplos es la categoría de grupos graduados. Veremos cómo especificar algebraicamente los grupos graduados y cómo trabajar con ellos. La motivación para incluir este ejemplo es nuestro interés en EAT y, por tanto, en los cálculos en Topología Algebraica, donde es muy habitual el trabajo con estructuras graduadas (estando la noción de graduación relacionada con la de dimensión geométrica). En EAT no se trabaja con grupos graduados, pero sí con estructuras graduadas más complejas. Así, nos ocupamos de este ejemplo porque, aun siendo sencillo, nos permitirá mostrar las peculiaridades de la modelización de este tipo de estructuras. El segundo ejemplo que vamos a tratar es la categoría de los conjuntos simpliciales. Este sí es uno de los casos reales con los que trabaja EAT. Con él pretendemos mostrar el tratamiento de la parcialidad de las operaciones e ilustrar en un caso concreto la necesidad de fijar los dominios de definición de los operadores, tal y como se ha hecho en el estudio teórico previamente realizado. El hecho de analizar en primer lugar la categoría de grupos graduados ayudará a la hora de abordar este segundo ejemplo ya que las dificultades que provienen de la graduación ya habrán sido tratadas en el primero. El tercer ejemplo también corresponde a uno de los casos reales con los que EAT trabaja y es la categoría de complejos de cadenas. Con él pretendemos mostrar cómo las propiedades matemáticas de los espacios y de sus elementos se trasladan a sus especificaciones y cómo de ahí es posible trasladarlas también a sus implementaciones.

2.9.1 Grupos Graduados

Un *grupo graduado* G es una aplicación que a cada $n \in \mathbb{Z}$ le asocia un grupo, que denotaremos por G_n. La forma más habitual de expresar un grupo graduado G es como una familia $G = \{G_n\}_{n\in\mathbb{Z}}$, siendo G_n un grupo para cada número entero n.

Un *morfismo entre grupos graduados* es una familia indexada por $\mathbb{Z}$ de homomorfismos de grupos, cada uno de los cuales es un homomorfismo entre los grupos correspondientes al mismo índice. Los grupos graduados junto con los morfismos entre ellos forman una categoría que llamaremos Categoría de Grupos Graduados y que denotaremos por *GrpGrd*.

Para modelar la categoría *GrpGrd* pretendemos definir un TAD para los grupos graduados, es decir, buscamos una signatura y una categoría de álgebras cuya relación con la categoría de grupos graduados quede bien establecida y tenga buenas propiedades.

Una primera observación es que pretendemos trabajar en el marco clásico de Especificación Algebraica (el desarrollado en [40] y [72], entre otros) el que, como mostraremos, resulta suficiente para formalizar todas las estructuras de datos utilizadas en EAT, sin recurrir, por ejemplo, a otros formalismos de orden superior (véanse como ejemplo [75], [82], [83], [19] y [5]).

Una primera posibilidad es tratar de describir la signatura que se obtiene directamente de la noción de grupo graduado, de la misma forma que se ha hecho con la de grupo. La diferencia entre ambas es que, en el caso que nos ocupa y en general en cualquier estructura graduada, aparece una cantidad numerable de géneros y operaciones. Además, si pensamos en especificaciones, también se tiene una cantidad numerable de axiomas.

Así, llamamos `GrpGrd` a la siguiente signatura:

- el conjunto de géneros es $\mathbb{Z}$;

- el conjunto de operaciones es $\Omega = \{prd_n\colon n\ n \to n, inv_n\colon n \to n, unt_n\colon \to n\}_{n\in\mathbb{Z}}$.

Llamamos $\mathcal{C}(\texttt{GrpGrd})$ a la subcategoría plena de $PAlg(\texttt{GrpGrd})$ cuyos objetos son las $\texttt{GrpGrd}$-álgebras totales $A = \langle \{A_n\}_{n\in\mathbb{Z}}, \{prd_{n_A}, inv_{n_A}, unt_{n_A}\}_{n\in\mathbb{Z}}\rangle$ verificando que, para todo $n \in \mathbb{Z}$, $\langle A_n, \{prd_{n_A}, inv_{n_A}, unt_{n_A}\}\rangle$ es grupo. La condición anterior se traduce en el cumplimiento de los siguientes axiomas:

$$\cdot\ n \in \mathbb{Z};\ x, y, z \in A_n \implies prd_{n_A}(prd_{n_A}(x, y), z) = prd_{n_A}(x, prd_{n_A}(y, z))$$

$$\cdot\ n \in \mathbb{Z};\ x \in A_n \implies prd_{n_A}(x, unt_{n_A}(*)) = prd_{n_A}(unt_{n_A}(*), x) = x$$

$$\cdot\ n \in \mathbb{Z};\ x \in A_n \implies prd_{n_A}(x, inv_{n_A}(x)) = prd_{n_A}(inv_{n_A}(x), x) = unt_{n_A}(*)$$

Es claro que $\mathcal{C}(\texttt{GrpGrd})$ es (isomorfa a) la categoría de Grupos Graduados.

Según se ha explicado en el capítulo, para acercarnos a la forma en que se trabajaría con los grupos graduados en EAT, debemos aplicar la construcción $()_{imp}$. Siguiendo la notación introducida en el capítulo, llamamos $\texttt{GrpGrd}_{imp}$ a la siguiente signatura:

- G_{imp}, el conjunto de géneros, está formado por $G_{imp} = \{imp_{\texttt{GrpGrd}}\} \cup \mathbb{Z}$;
- el conjunto de operaciones consta de $\Omega_{imp} = \{imp\_prd_n\colon imp_{\texttt{GrpGrd}}\ n\ n \to n,$ $imp\_inv_n\colon imp_{\texttt{GrpGrd}}\ n \to n,\ imp\_unt_n\colon imp_{\texttt{GrpGrd}} \to n\}_{n\in\mathbb{Z}}$

y, consideramos la subcategoría plena de $PAlg(\texttt{GrpGrd}_{imp})$ que denotamos por $\mathcal{C}(\texttt{GrpGrd}_{imp})$, cuyos objetos son las $\texttt{GrpGrd}_{imp}$-álgebras tales que la $\texttt{GrpGrd}$-álgebra determinada por cada elemento del soporte correspondiente a $imp_{\texttt{GrpGrd}}$ está en $\mathcal{C}(\texttt{GrpGrd})$.

Como ha quedado claro a lo largo del capítulo, nuestro interés se centra en encontrar $\texttt{GrpGrd}_{imp}$-álgebras lo más generales posible que, de alguna manera, recojan las $\texttt{GrpGrd}$-álgebras de partida, ya que es una de las formas de poder trabajar con "cualquier" grupo graduado. Por eso, nuestro objetivo ha sido localizar objetos finales que poseen propiedades de generalidad, en el sentido de que desde cualquier otro objeto, siempre hay un paso natural (morfismo canónico) a ellos. A nivel teórico, hemos encontrado una familia de objetos (uno por cada conjunto de soportes que se fije) con las características deseadas. La descripción que hemos dado de cada uno de estos objetos tiene como soporte para el género imp_Σ un conjunto de tuplas de funciones, cada una de las cuales está formada por tantas funciones como operadores tenga la signatura de partida. Ahora bien, la noción de tupla lleva consigo el poseer un número finito de componentes, así que, en el caso de una signatura con infinitas operaciones como la que ahora nos ocupa, la expresión del objeto final no vendría dada por tuplas sino por un producto infinito de espacios de funciones. Continuar el estudio por esta vía lleva consigo el trabajar con productos infinitos lo que, algebraicamente, no supone problema alguno. Sin embargo, el problema se plantea al intentar reproducirlo en las implementaciones, donde encontramos dificultades por la infinitud. En particular, ya en la máquina, nos encontraríamos con el problema de dar la representación natural en Lisp del objeto final. (Las estructuras apropiadas para la representación de productos de espacios de funciones son registros, y éstos tienen un número finito de campos.) Así, optamos por no continuar por esta vía nuestro estudio de los grupos graduados, siendo que, además, no es así como se trabaja con ellos en EAT.

Pensando en alternativas para especificar las estructuras graduadas, podemos ver que lo que sí hay es una especie de "patrón" común a todas ellas. Esto sugiere que una posibilidad

sería admitir variables de género (también de operaciones y de axiomas), es decir, variables que cuantificasen el conjunto de los géneros (en las especificaciones habituales se admiten variables pero que se refieren a elementos de los soportes). Hacerlo así supone un cambio de paradigma ya que estaríamos saliéndonos de la Especificación Algebraica clásica, marco que no queremos abandonar, por lo que tampoco abordamos el análisis por esta vía ([75], [82], [83], [19] y [5] son ejemplos de trabajos no enmarcados en la Especificación Algebraica clásica).

Nos planteamos la cuestión de por qué este ejemplo es diferente de los vistos hasta aquí. La principal diferencia está en que en los ejemplos tratados hasta ahora aparecen como nociones esenciales la de elemento y la de operación entre elementos, y la Especificación Algebraica clásica es una buena herramienta para modelar las estructuras que provienen del Álgebra Universal, basadas fundamentalmente en dichas nociones. Sin embargo, no puede decirse, en rigor, que un grupo graduado posea elementos ni operaciones: se trata de una función, y son los *elementos* de la imagen los que son grupos (son éstos los que constan de elementos y operaciones entre ellos). Vamos ahora a intentar aproximarnos a la noción de grupo graduado de las Matemáticas utilizando signaturas que posean solo un número finito de géneros y de operaciones. Para ello, vamos a considerar un género que denotamos por elm. La idea intuitiva es que en el soporte correspondiente a elm se encuentren todos los elementos de cada uno de los grupos de los que consta el grupo graduado. Aunque un grupo graduado no posee operaciones, en cada uno de los grupos que lo componen sí hay operaciones, así en la signatura que vamos a considerar incluimos tres operaciones que abstraen las operaciones internas en cada uno de los grupos. La parcialidad permitirá utilizar esta representación en el caso del producto, pero en el caso del elemento neutro, será necesario distinguir el de cada uno de los grupos, por lo que deberemos asociar cada uno con un entero (su grado).

Concretando, consideramos la signatura que denotamos por `GG`, compuesta por dos géneros, uno que llamamos elm, y otro asociado a la graduación del grupo que denotamos por int, con las siguientes operaciones:

$$\begin{array}{lcll} gprd & : & elm\ \ elm & \rightarrow\ elm \\ ginv & : & elm & \rightarrow\ elm \\ gunt & : & int & \rightarrow\ elm \end{array}$$

Con las operaciones anteriores pretendemos recoger las operaciones internas en cada grupo, es decir:

- el producto de dos elementos pertenecientes al mismo grupo;
- el inverso de los elementos en un grupo;
- y, por último, una operación constante que es el cálculo del elemento neutro en cada grupo.

Para obtener una categoría de álgebras que corresponda con la categoría de grupos graduados, es claro que no debemos considerar todas las `GG`-álgebras. Consideramos una subcategoría de $PAlg(\texttt{GG})$ que denominamos $\mathcal{C}(\texttt{GG})$ y que se define como la subcategoría de $PAlg(\texttt{GG})$ con objetos las `GG`-álgebras A que verifican las siguientes condiciones:

i) El soporte para el género int es $\mathbb{Z}$.

ii) Existe un predicado $grado_A \subseteq \mathbb{Z} \times A_{elm}$. La pertenencia de un par (x, n) a $grado_A$ se interpreta diciendo que x es considerado de grado n. Esta relación será denotada por $grado_A(x, n)$ en lo que sigue.

iii) La función $gprd_A$ es parcial con dominio de definición:

$$Def(gprd_A) = \big\{(x, y) \in A_{elm} \times A_{elm} \mid \exists n \in \mathbb{Z} \text{ con } grado_A(x, n) \text{ y } grado_A(y, n)\big\}$$

iv) Las funciones $ginv_A$ y $gunt_A$ son totales.

v) Las operaciones tienen que ser compatibles con el predicado $grado_A$. Es decir, debe verificarse lo siguiente:

- para cada $n \in \mathbb{Z}$ se tiene que $grado_A(gunt_A(n), n)$;
- para cada $x \in A_{elm}$ y cada $n \in \mathbb{Z}$ tales que $grado_A(x, n)$, se tiene que $grado_A(ginv_A(x), n)$;
- para cada par $x, y \in A_{elm}$ y cada $n \in \mathbb{Z}$ tales que $grado_A(x, n)$ y $grado_A(y, n)$, se tiene que $grado_A(gprd_A(x, y), n)$.

vi) En cada grado lo que se quiere representar es un grupo, por tanto debe verificarse que, para cada $n \in \mathbb{Z}$, el conjunto $X_n = \big\{x \in A_{elm} \mid grado_A(x, n)\big\}$, junto con las operaciones $gprd_A|_{X_n \times X_n}$, $ginv_A|_{X_n}$ y $gunt_A|_{\{n\}}$ debe ser un grupo. Es decir, debe cumplirse lo siguiente:

- para cada terna $x, y, z \in A_{elm}$ y cada $n \in \mathbb{Z}$ con $grado_A(x, n)$, $grado_A(y, n)$ y $grado_A(z, n)$, se tiene que $gprd_A(gprd_A(x, y), z) = gprd_A(x, gprd_A(y, z))$;
- para cada $x \in A_{elm}$ y cada $n \in \mathbb{Z}$ tales que $grado_A(x, n)$, se tiene que $gprd_A(x, gunt_A(n)) = gprd_A(gunt_A(n), x) = x$;
- para cada $x \in A_{elm}$ y cada $n \in \mathbb{Z}$ tales que $grado_A(x, n)$, se tiene que $gprd_A(x, ginv_A(x)) = gprd_A(ginv_A(x), x) = gunt_A(n)$.

Como morfismos de $\mathcal{C}(\texttt{GG})$ consideramos aquellos que son la identidad sobre $\mathbb{Z}$ y que respetan el grado de los elementos.

Estudiamos la relación entre la categoría de grupos graduados, *GrpGrd*, y la categoría $\mathcal{C}(\texttt{GG})$.

Cada objeto de $\mathcal{C}(\texttt{GG})$ determina de manera natural un grupo graduado. Si $A = \big\langle A_{elm}, \mathbb{Z}, \{(gprd_A, Def(gprd_A)), (ginv_A, A_{elm}), (gunt_A, \mathbb{Z})\}\big\rangle$ es objeto de $\mathcal{C}(\texttt{GG})$, A determina el grupo graduado $G = \{G_n\}_{n \in \mathbb{Z}}$ siendo, para cada $n \in \mathbb{Z}$, $G_n = \big\langle X_n, \{prd_{G_n}, inv_{G_n}, unt_{G_n}\}\big\rangle$ el grupo sobre el conjunto $X_n = \{x \in A_{elm} \mid grado_A(x, n)\}$ con las operaciones definidas de forma inmediata a partir de las de A. La propiedad recogida en *vi*) garantiza que, para todo $n \in \mathbb{Z}$, G_n es grupo. Como además, mediante la restricción a cada grupo, los morfismos entre objetos de $\mathcal{C}(\texttt{GG})$ inducen morfismos entre grupos graduados, lo anterior define un funtor $H : \mathcal{C}(\texttt{GG}) \to GrpGrd$.

Sin embargo, no es tan claro cómo determinar, a partir de un grupo graduado, un álgebra de $\mathcal{C}(\texttt{GG})$. Nos planteamos en primer lugar, cómo determinar el soporte para el género *elm*. En este sentido, parece natural considerar el conjunto unión de los elementos de todos los grupos, y tomarlo como soporte para el género *elm*. Así, no solo podemos asegurar la existencia de un predicado *grado*, sino que incluso podemos definirlo: un elemento del soporte asociado a *elm* es de grado n si y solo si el elemento está en el grupo cuyo índice es n. Sin embargo,

la definición de las operaciones nos puede plantear problemas. Mostramos a continuación un ejemplo en el que se puede definir, tal y como acabamos de describir, un objeto de $\mathcal{C}(\texttt{GG})$ a partir de un grupo graduado. Consideramos el grupo graduado formado por copias de $\mathbb{Z}$, $\{\mathbb{Z}\}_{n\in\mathbb{Z}}$. La GG-álgebra A de $\mathcal{C}(\texttt{GG})$ determinada a partir del grupo graduado anterior tal y como se acaba de explicar, es la que tiene a $\mathbb{Z}$ como soporte para *elm* y como predicado $grado_A$ al mayor de los posibles, es decir, cualquier elemento de $\mathbb{Z}$ puede ser considerado de cualquier grado. A partir de esto, la definición de las operaciones es evidente: $gprd_A(z_1, z_2) := z_1 + z_2$, $ginv_A(z_1) := -z_1$ y $gunt_A(n) := 0$, para todo $z_1, z_2, n \in \mathbb{Z}$. En este caso, las operaciones de los grupos del grupo graduado son iguales para todos los $n \in \mathbb{Z}$, lo que hace que la definición anterior sea correcta. Sin embargo, puede ocurrir que las operaciones no estén definidas de forma compatible sobre elementos que puedan ser considerados de más de un grado, por lo que lo que se determina del modo anterior no siempre es un álgebra de $\mathcal{C}(\texttt{GG})$.

Una forma de solucionar lo anterior es tomar como soporte para *elm* la unión disjunta de los conjuntos base de todos los grupos. Así, tomamos como soporte para *elm* el conjunto $\bigcup_{n\in\mathbb{Z}}(\{n\} \times X_n)$, siendo X_n el conjunto base del grupo G_n, para cada $n \in \mathbb{Z}$. Los elementos del conjunto anterior los expresaremos como pares $< n, x >$ con $n \in \mathbb{Z}$ y $x \in X_n$. El predicado *grado* se define del siguiente modo: $grado_A(< n, x >, i)$ si y solo si $i = n$. Es claro que, en este caso, el predicado *grado* puede verse como una función $grado_A \colon A_{elm} \to \mathbb{Z}$.

Por ejemplo, volviendo al grupo graduado $\{\mathbb{Z}\}_{n\in\mathbb{Z}}$, construimos un álgebra B de $\mathcal{C}(\texttt{GG})$ tomando como soporte para *elm* el conjunto $\mathbb{Z} \times \mathbb{Z}$ (biyectivo con la unión disjunta de los conjuntos base de los grupos), siendo la primera componente la que determina el grado del elemento de la segunda. Las operaciones se definen naturalmente.

La observación esencial en lo anterior es que si el predicado *grado* de una GG-álgebra define una función $grado \colon elm \to int$, entonces el soporte A_{elm} puede verse como la unión disjunta $\bigsqcup_{n\in\mathbb{Z}} grado_A^{-1}(n)$ siendo, para cada $n \in \mathbb{Z}$, $grado_A^{-1}(n)$ el conjunto preimagen de n por la función *grado*. Esto va a dar una clase de GG-álgebras que se puede identificar con los grupos graduados.

De esta forma podemos considerar otra categoría que denominamos $\tilde{\mathcal{C}}(\texttt{GG})$ y cuyos objetos se diferencian esencialmente de los de $\mathcal{C}(\texttt{GG})$ en la existencia de una función *grado* (en lugar de un predicado), lo que lleva consigo la modificación de algunas de las condiciones exigidas a las álgebras de $\mathcal{C}(\texttt{GG})$. Así, las condiciones que se exigen a una GG-álgebra A de $\tilde{\mathcal{C}}(\texttt{GG})$ son las siguientes:

i) El soporte para el género *int* es $\mathbb{Z}$.

ii′) Existe una función $grado_A \colon A_{elm} \to \mathbb{Z}$ que calcula el grado de cada elemento de A_{elm}.

iii′) El dominio de definición de la función $gprd_A$ es el siguiente:

$$Def(gprd_A) = \big\{(x, y) \in A_{elm} \times A_{elm} \;\mid\; grado_A(x) = grado_A(y)\big\}$$

iv) Las funciones $ginv_A$ y $gunt_A$ son totales.

v′) La función $grado_A$ tiene que ser compatible con las operaciones, es decir, debe verificar lo siguiente:

- para cada $n \in \mathbb{Z}$, $grado_A(gunt_A(n)) = n$;
- para cada $x \in A_{elm}$ y cada $n \in \mathbb{Z}$ tales que $grado_A(x) = n$, se tiene que $grado_A(ginv_A(x)) = n$;

- para cada par $x, y \in A_{elm}$ y cada $n \in \mathbb{Z}$ tales que $grado_A(x) = n$ y $grado_A(y) = n$, se tiene que $grado_A(gprd_A(x, y)) = n$.

vi') En cada grado debe tenerse un grupo, por tanto debe verificarse que, para cada $n \in \mathbb{Z}$, el conjunto $X_n = \{x \in A_{elm} \mid grado_A(x) = n\}$, junto con las operaciones $gprd_A|_{X_n \times X_n}$, $ginv_A|_{X_n}$ y $gunt_A|_{\{n\}}$ debe tener estructura de grupo, es decir, deben verificarse las siguientes condiciones:

- para cada terna $x, y, z \in A_{elm}$ y cada $n \in \mathbb{Z}$ tales que $grado_A(x) = n$, $grado_A(y) = n$ y $grado_A(z) = n$, se tiene que $gprd_A(gprd_A(x, y), z) = gprd_A(x, gprd_A(y, z))$;
- para cada $x \in A_{elm}$ y cada $n \in \mathbb{Z}$ tales que $grado_A(x) = n$, se tiene que $gprd_A(x, gunt_A(n)) = gprd_A(gunt_A(n), x) = x$;
- para cada $x \in A_{elm}$ y cada $n \in \mathbb{Z}$ tales que $grado_A(x) = n$, se tiene que $gprd_A(x, ginv_A(x)) = gprd_A(ginv_A(x), x) = gunt_A(n)$.

Como ya se ha comentado, la existencia de la función $grado_A$ da una descomposición de A_{elm} como la unión disjunta con índices en $\mathbb{Z}$ de las preimágenes de dicha función. Así, sin perder generalidad, siempre usaremos descripciones de A_{elm} como unión disjunta con índices en $\mathbb{Z}$, pudiendo tomar distintas representaciones para ésta. Respecto de los morfismos, basta imponer que sean la identidad sobre $\mathbb{Z}$, ya que, en este caso, esto garantiza que conservan el grado.

Observar que todo objeto de $\tilde{\mathcal{C}}(\mathtt{GG})$ es también objeto de $\mathcal{C}(\mathtt{GG})$ (toda función define un predicado) y, por tanto, que cada objeto de $\tilde{\mathcal{C}}(\mathtt{GG})$ determina un grupo graduado. Abusando de notación, volvemos a denotar por H a la restricción a $\tilde{\mathcal{C}}(\mathtt{GG})$ del funtor definido con anterioridad, es decir, consideramos el funtor $H : \tilde{\mathcal{C}}(\mathtt{GG}) \to GrpGrd$.

Inversamente, cada grupo graduado determina un álgebra de $\tilde{\mathcal{C}}(\mathtt{GG})$ y cada morfismo entre grupos graduados induce un morfismo en $\tilde{\mathcal{C}}(\mathtt{GG})$. Formalmente, esto corresponde a la existencia de un funtor L: $GrpGrd \to \tilde{\mathcal{C}}(\mathtt{GG})$ que asocia, a cada grupo graduado $G = \{G_n\}_{n\in\mathbb{Z}}$ con $G_n = \langle X_n, \{prd_n, inv_n, unt_n\}\rangle$, una $\mathtt{GG}$-álgebra $L(G) := A$, siendo $A = \langle A_{elm}, \mathbb{Z}, \{(gprd_A, Def(gprd_A)), (ginv_A, A_{elm}), (gunt_A, \mathbb{Z})\}\rangle$ la $\mathtt{GG}$-álgebra definida por: $A_{elm} = \bigcup_{n\in\mathbb{Z}}(\{n\} \times X_n)$, $Def(gprd_A) = \{(< n, x >, < m, y >) \in A_{elm} \times A_{elm} \mid n = m\}$ y con operaciones $gprd_A(< n, x >, < n, y >) :=< n, prd_n(x, y) >$, $ginv_A(< n, x >) :=< n, inv_n(x) >$ y $gunt_A(n) :=< n, unt_n(*) >$. Con la definición anterior no solo está asegurada la existencia de una función $grado_A$, sino que podemos definirla; $grado_A(< n, x >) := n$, para todo $< n, x >\in A_{elm}$. Así, A verifica las condiciones recogidas en *vi*).

Los funtores H y L definen una equivalencia de categorías entre $\tilde{\mathcal{C}}(\mathtt{GG})$ y la categoría de grupos graduados, por lo que podemos establecer que ambas categorías tienen el mismo poder expresivo.

Lo anterior nos lleva a considerar el TAD $\mathcal{T}_{\mathtt{GG}} = \langle \mathtt{GG}, \tilde{\mathcal{C}}(\mathtt{GG})\rangle$ que, por la equivalencia anterior, denominamos *TAD de los Grupos Graduados*.

Es importante señalar que en todo el capítulo hemos trabajado con signaturas y no con especificaciones y para obtener una subcategoría de álgebras lo hemos hecho exigiendo propiedades a las álgebras (es decir, estamos utilizando una aproximación semántica basada en modelos; ver en preliminares, página 27, los distintos tipos de aproximaciones semánticas). En este ejemplo concreto puede verse que realmente es indistinto trabajar como lo hemos hecho o mediante el establecimiento de axiomas, ya que todas las condiciones exigidas pueden escribirse axiomáticamente. Para trabajar axiomáticamente en este caso, el marco adecuado sería

el marco oculto ya que el hecho de fijar $\mathbb{Z}$ como conjunto soporte para el género *int* tiene una interpretación clara en el marco oculto: fijar el álgebra de datos. La interpretación en el marco clásico de especificación no es tan clara.

El siguiente paso es considerar el TAD $\mathcal{T}_{\mathtt{GG}_{imp}} = \langle \mathtt{GG}_{imp}, \tilde{\mathcal{C}}(\mathtt{GG}_{imp}) \rangle$. Para ello, construimos en primer lugar la signatura $\mathtt{GG}_{imp}$, que además de los géneros *int* y *elm* de la signatura GG, incluye un nuevo género $imp_{\mathtt{GG}}$. Las operaciones son las siguientes:

$$\begin{array}{lcllll} imp\_gprd & : & imp_{\mathtt{GG}} & elm & elm & \rightarrow & elm \\ imp\_ginv & : & imp_{\mathtt{GG}} & elm & & \rightarrow & elm \\ imp\_gunt & : & imp_{\mathtt{GG}} & int & & \rightarrow & elm \end{array}$$

Por su parte, $\tilde{\mathcal{C}}(\mathtt{GG}_{imp})$ será la subcategoría plena de $PAlg(\mathtt{GG}_{imp})$ con objetos las $\mathtt{GG}_{imp}$-álgebras $A = \langle A_{imp_{\mathtt{GG}}}, A_{elm}, \mathbb{Z}, \{(imp\_gprd_A, Def(imp\_gprd_A)), (imp\_ginv_A, Def(imp\_ginv_A)), (imp\_gunt_A, Def(imp\_gunt_A))\}\rangle$, tales que, para todo $a \in A_{imp_{\mathtt{GG}}}$, la GG-álgebra $A_a = \langle A_{elm}, \mathbb{Z}, \{(imp\_gprd_A(a,-), Def(imp\_gprd_A(a,-))), (imp\_ginv_A(a,-), Def(imp\_ginv_A(a,-))), (imp\_gunt_A(a,-), Def(imp\_gunt_A(a,-)))\}\rangle$ está en $\tilde{\mathcal{C}}(\mathtt{GG})$. Teniendo en cuenta que cada elemento de $\tilde{\mathcal{C}}(\mathtt{GG})$ determina un grupo graduado, cada objeto de $\tilde{\mathcal{C}}(\mathtt{GG}_{imp})$ representa a una familia indexada de grupos graduados.

Como se ha visto en el desarrollo teórico realizado, para obtener $\mathtt{GG}_{imp}$-álgebras con buenas propiedades de generalidad es necesario fijar los soportes correspondientes a los géneros de la signatura GG. Desde el punto de vista práctico, fijar los soportes viene motivado por el hecho de necesitar un patrón común para los elementos de todas las estructuras que van a manipularse, ya que hay que poder aplicarles determinadas operaciones por lo que los elementos deberán tener un aspecto "sintáctico" parecido. Por ello, fijamos un universo $\mathcal{U}$ suficientemente rico y restringimos las categorías $\tilde{\mathcal{C}}(\mathtt{GG})$ y *GrpGrd* tomando los soportes de las álgebras y los conjuntos base de los grupos dentro de $\mathcal{U}$. En este caso particular, fijar el soporte para *elm* supone elegir una representación para una reunión disjunta con índices en $\mathbb{Z}$. Para hacer esto explícito los elementos del soporte de género *elm* siempre los escribiremos como pares $< n, x >$ con $n \in \mathbb{Z}$, siendo la función *grado* la proyección en la primera componente. Así, bastará con señalar un conjunto D, de forma que el soporte sea un conjunto de la forma $\mathbb{Z} \times D$ y considerar funciones parciales. Así, la definición de la función *grado* será $grado(< n, x >) := n$, para todo par $< n, x >$ del soporte.

Sin embargo, una forma habitual de trabajar consiste en no fijar un conjunto como soporte, sino generar los elementos de A_{elm} a partir de determinados elementos básicos. De hecho, es así como realmente se trabaja en EAT. Dicho de otro modo, conviene contemplar el soporte para *elm* como un álgebra inicial para alguna signatura concreta. Las operaciones necesarias para generar los elementos de los soportes de la signatura de partida que deben fijarse, no son operaciones de las estructuras algebraicas a manipular, por lo que, desde el punto de vista de las Especificaciones Ocultas, son operaciones visibles. Ésta es una de las causas por las que pueden aparecer operaciones visibles, la necesidad de generar los elementos de los soportes correspondientes a los géneros visibles, es decir, los elementos del álgebra de datos.

Presentamos a continuación algunos ejemplos concretos de $\mathtt{GG}_{imp}$-álgebras. Comenzamos por fijar el soporte correspondiente a *elm*, que será de la forma $\mathbb{Z} \times D$. Tomamos $D = \mathbb{Z}$. Siguiendo la misma notación que en el resto del capítulo, denotamos por $\tilde{\mathcal{C}}^{\mathbb{Z}\times\mathbb{Z}}(\mathtt{GG}_{imp})$ a la

subcategoría plena de $\tilde{\mathcal{C}}(\mathtt{GG}_{imp})$ cuyos objetos tienen a $\mathbb{Z}\times\mathbb{Z}$ como soporte para el género *elm* y con función *grado* la proyección en la primera componente. Vamos a dar algunas álgebras de $\tilde{\mathcal{C}}^{\mathbb{Z}\times\mathbb{Z}}(\mathtt{GG}_{imp})$. Todos los ejemplos que mostramos a continuación tienen a $\mathbb{N}$ como soporte para el género distinguido $imp_{\mathtt{GG}}$:

1) Definimos una $\mathtt{GG}_{imp}$-álgebra que representa a una familia de grupos graduados, todos ellos distintos, pero de forma que cada grupo graduado se compone del mismo grupo en todos los grados. Consideramos la $\mathtt{GG}_{imp}$-álgebra A con operaciones definidas del siguiente modo:

   · $imp\_gprd_A\colon \mathbb{N}\times(\mathbb{Z}\times\mathbb{Z})\times(\mathbb{Z}\times\mathbb{Z}) \to \mathbb{Z}\times\mathbb{Z}$ función parcial con dominio $Def(imp\_gprd_A) = \{(n,<i_1,x_1>,<i_2,x_2>)\in\mathbb{N}\times(\mathbb{Z}\times\mathbb{Z})\times(\mathbb{Z}\times\mathbb{Z}) \mid i_1 = i_2\}$ definida por $imp\_gprd_A(n,<i,x_1>,<i,x_2>) := <i, x_1+x_2-n>$, para todo $(n,<i,x_1>,<i,x_2>)\in Def(imp\_gprd_A)$;

   · $imp\_ginv_A\colon \mathbb{N}\times(\mathbb{Z}\times\mathbb{Z}) \to \mathbb{Z}\times\mathbb{Z}$ función total definida por $imp\_ginv_A(n,<i,x>) := <i, 2*n-x>$, e

   · $imp\_gunt_A\colon \mathbb{N}\times\mathbb{Z}\to\mathbb{Z}\times\mathbb{Z}$ función total definida por $imp\_gunt_A(n,i) := <i,n>$.

   La $\mathtt{GG}_{imp}$-álgebra A está en $\tilde{\mathcal{C}}^{\mathbb{Z}\times\mathbb{Z}}(\mathtt{GG}_{imp})$ y representa a la familia de grupos graduados $\{A_n\}_{n\in\mathbb{N}}$ donde, para cada $n\in\mathbb{N}$, A_n es un grupo graduado isomorfo al formado por copias del grupo trasladado de $\mathbb{Z}$, $\vec{\mathbb{Z}}^n$, es decir, $A_n \cong \{\vec{\mathbb{Z}}^n\}_{i\in\mathbb{Z}}$, siendo $\vec{\mathbb{Z}}^n = \langle\mathbb{Z},\{prd_n, inv_n, unt_n\}\rangle$ con $prd_n(z_1,z_2) := z_1+z_2-n$, $inv_n(z_1) := 2*n-z_1$ y $unt_n(*) := n$, para todo $z_1,z_2\in\mathbb{Z}$.

2) La $\mathtt{GG}_{imp}$-álgebra B que definimos a continuación representa a la familia de grupos graduados indexada por los naturales en la que cada elemento de la familia es un grupo graduado isomorfo a $B_n = \{\vec{\mathbb{Z}}^i\}_{i\in\mathbb{Z}}$, por tanto, la familia está formada por un único grupo graduado. Las operaciones de B se definen como sigue:

   · $imp\_gprd_B\colon \mathbb{N}\times(\mathbb{Z}\times\mathbb{Z})\times(\mathbb{Z}\times\mathbb{Z}) \to \mathbb{Z}\times\mathbb{Z}$ con $Def(imp\_gprd_B) = \{(n,<i_1,x_1>,<i_2,x_2>)\in\mathbb{N}\times(\mathbb{Z}\times\mathbb{Z})\times(\mathbb{Z}\times\mathbb{Z}) \mid i_1 = i_2\}$ definida por $imp\_gprd_B(n,<i,x_1>,<i,x_2>) := <i, x_1+x_2-i>$, para todo $(n,<i,x_1>,<i,x_2>)\in Def(imp\_gprd_B)$;

   · $imp\_ginv_B\colon \mathbb{N}\times(\mathbb{Z}\times\mathbb{Z}) \to \mathbb{Z}\times\mathbb{Z}$ es la función total definida por $imp\_ginv_B(n,<i,x>) := <i, 2*i-x>$; y, por último,

   · $imp\_gunt_B\colon \mathbb{N}\times\mathbb{Z}\to\mathbb{Z}\times\mathbb{Z}$ es la función total definida por $imp\_gunt_B(n,i) := <i,i>$.

3) Definimos a continuación una $\mathtt{GG}_{imp}$-álgebra que representa a una familia de grupos graduados, todos ellos distintos y conteniendo grupos distintos en cada grado. Sea C la $\mathtt{GG}_{imp}$-álgebra de $\tilde{\mathcal{C}}^{\mathbb{Z}\times\mathbb{Z}}(\mathtt{GG}_{imp})$ con las siguientes operaciones:

   · $imp\_gprd_C\colon \mathbb{N}\times(\mathbb{Z}\times\mathbb{Z})\times(\mathbb{Z}\times\mathbb{Z})\to\mathbb{Z}\times\mathbb{Z}$ función parcial con $Def(imp\_gprd_C) = \{(n,<i_1,x_1>,<i_2,x_2>)\in\mathbb{N}\times(\mathbb{Z}\times\mathbb{Z})\times(\mathbb{Z}\times\mathbb{Z}) \mid i_1 = i_2\}$ definida por $imp\_gprd_C(n,<i,x_1>,<i,x_2>) := <i, x_1+x_2-(n+i)>$, para todo $(n,<i,x_1>,<i,x_2>)\in Def(imp\_gprd_C)$;

   · $imp\_ginv_C\colon \mathbb{N}\times(\mathbb{Z}\times\mathbb{Z}) \to \mathbb{Z}\times\mathbb{Z}$ función total definida por $imp\_ginv_C(n,<i,x>) := <i, 2*(n+i)-x>$; y, finalmente,

   · $imp\_gunt_C\colon \mathbb{N}\times\mathbb{Z}\to\mathbb{Z}\times\mathbb{Z}$ función total definida por $imp\_gunt_C(n,i) := <i, n+i>$.

Esta $\mathtt{GG}_{imp}$-álgebra representa a la familia de grupos graduados $\{C_n\}_{n\in\mathbb{N}}$ siendo, para cada $n \in \mathbb{N}$, C_n un grupo graduado isomorfo al grupo graduado $\{\mathbb{Z}^{\rightarrow(n+i)}\}_{i\in\mathbb{Z}}$.

Siguiendo la misma notación que en el resto del capítulo, denotamos por $\tilde{\mathcal{C}}^{\mathbb{Z}\times\mathbb{Z},\{\}}(\mathtt{GG}_{imp})$ a la subcategoría de $\tilde{\mathcal{C}}^{\mathbb{Z}\times\mathbb{Z}}(\mathtt{GG}_{imp})$ cuyos morfismos son la identidad sobre los géneros de GG. De acuerdo con la teoría, la categoría $\tilde{\mathcal{C}}^{\mathbb{Z}\times\mathbb{Z},\{\}}(\mathtt{GG}_{imp})$ posee objeto final, objeto que denotamos por $F^{\mathbb{Z}\times\mathbb{Z}}$ y cuya descripción damos a continuación:

- Como soporte para el género distinguido se toma el siguiente espacio de funciones, definido a partir de los objetos de $\tilde{\mathcal{C}}(\mathtt{GG})$:

$$F^{\mathbb{Z}\times\mathbb{Z}}_{imp_{\mathtt{GG}}} = \big\{ f = \big(f_{gprd}\colon (\mathbb{Z}\times\mathbb{Z})\times(\mathbb{Z}\times\mathbb{Z}) \to \mathbb{Z}\times\mathbb{Z}, f_{ginv}\colon \mathbb{Z}\times\mathbb{Z} \to \mathbb{Z}\times\mathbb{Z}, f_{gunt}\colon \mathbb{Z} \to \mathbb{Z}\times\mathbb{Z}\big) \mid$$

$$\mid \big\langle \mathbb{Z}\times\mathbb{Z}, \mathbb{Z}, \{(f_{gprd}, Def(f_{gprd})), (f_{ginv}, \mathbb{Z}\times\mathbb{Z}), (f_{gunt}, \mathbb{Z})\}\big\rangle \text{ es objeto de } \tilde{\mathcal{C}}(\mathtt{GG}) \ \big\}$$

  De esta forma el objeto de $\tilde{\mathcal{C}}(\mathtt{GG})$ determinado por cada tupla de funciones $f \in F^{\mathbb{Z}\times\mathbb{Z}}_{imp_{\mathtt{GG}}}$ define un grupo graduado.

- Las operaciones se definen como la evaluación en la componente correspondiente:
  - $imp\_gprd_{F^{\mathbb{Z}\times\mathbb{Z}}}\colon F^{\mathbb{Z}\times\mathbb{Z}}_{imp_{\mathtt{GG}}} \times (\mathbb{Z}\times\mathbb{Z}) \times (\mathbb{Z}\times\mathbb{Z}) \to \mathbb{Z}\times\mathbb{Z}$ función parcial con $Def(imp\_gprd_{F^{\mathbb{Z}\times\mathbb{Z}}}) = \big\{(f,a,b) \in F^{\mathbb{Z}\times\mathbb{Z}}_{imp_{\mathtt{GG}}} \times (\mathbb{Z}\times\mathbb{Z}) \times (\mathbb{Z}\times\mathbb{Z}) \mid (a,b) \in Def(f_{gprd})\big\}$ y definida por $imp\_gprd_{F^{\mathbb{Z}\times\mathbb{Z}}}(f,a,b) := f_{gprd}(a,b)$;
  - $imp\_ginv_{F^{\mathbb{Z}\times\mathbb{Z}}}\colon F^{\mathbb{Z}\times\mathbb{Z}}_{imp_{\mathtt{GG}}} \times (\mathbb{Z}\times\mathbb{Z}) \to \mathbb{Z}\times\mathbb{Z}$ función parcial con dominio $\big\{(f,a) \in F^{\mathbb{Z}\times\mathbb{Z}}_{imp_{\mathtt{GG}}} \times (\mathbb{Z}\times\mathbb{Z}) \mid a \in Def(f_{ginv})\big\}$ y definida por $imp\_ginv_{F^{\mathbb{Z}\times\mathbb{Z}}}(f,a) := f_{ginv}(a)$; y, por último,
  - $imp\_gunt_{F^{\mathbb{Z}\times\mathbb{Z}}}\colon F^{\mathbb{Z}\times\mathbb{Z}}_{imp_{\mathtt{GG}}} \times \mathbb{Z} \to \mathbb{Z}\times\mathbb{Z}$ función dada por $imp\_gunt_{F^{\mathbb{Z}\times\mathbb{Z}}}(f,n) := f_{gunt}(n)$ con $Def(imp\_gunt_{F^{\mathbb{Z}\times\mathbb{Z}}}) = \big\{(f,n) \in F^{\mathbb{Z}\times\mathbb{Z}}_{imp_{\mathtt{GG}}} \times \mathbb{Z} \mid n \in Def(f_{gunt})\big\}$ que, por definición de f_{gunt}, coincide con $F^{\mathbb{Z}\times\mathbb{Z}}_{imp_{\mathtt{GG}}} \times \mathbb{Z}$, por lo que la función $imp\_gunt_{F^{\mathbb{Z}\times\mathbb{Z}}}$ es total.

Los morfismos canónicos en $\tilde{\mathcal{C}}^{\mathbb{Z}\times\mathbb{Z},\{\}}(\mathtt{GG}_{imp})$, desde cada una de las tres $\mathtt{GG}_{imp}$-álgebras de $\tilde{\mathcal{C}}^{\mathbb{Z}\times\mathbb{Z}}(\mathtt{GG}_{imp})$ que hemos dado, sobre el objeto final $F^{\mathbb{Z}\times\mathbb{Z}}$ son, respectivamente:

1) En la componente $imp_{\mathtt{GG}}$, la aplicación $h_{A_{imp_{\mathtt{GG}}}}\colon \mathbb{N} \to F^{\mathbb{Z}\times\mathbb{Z}}_{imp_{\mathtt{GG}}}$ viene dada, para cada $n \in \mathbb{N}$, por $h_{A_{imp_{\mathtt{GG}}}}(n) := f_n$ con $f_n = (f_{n,gprd}, f_{n,ginv}, f_{n,gunt})$, funciones definidas por:
   - $f_{n,gprd}\colon (\mathbb{Z}\times\mathbb{Z})\times(\mathbb{Z}\times\mathbb{Z}) \to \mathbb{Z}\times\mathbb{Z}$ función parcial con dominio $Def(f_{n,gprd}) = \big\{(< i_1, x_1 >, < i_2, x_2 >) \in (\mathbb{Z}\times\mathbb{Z})\times(\mathbb{Z}\times\mathbb{Z}) \mid i_1 = i_2\big\}$ y definida por $f_{n,gprd}(< i, x_1 >, < i, x_2 >) := < i, x_1 + x_2 - n >$;
   - $f_{n,ginv}\colon \mathbb{Z}\times\mathbb{Z} \to \mathbb{Z}\times\mathbb{Z}$ función total dada por $f_{n,ginv}(< i, x >) := < i, 2*n - x >$; y,
   - $f_{n,gunt}\colon \mathbb{Z} \to \mathbb{Z}\times\mathbb{Z}$ función total definida por $f_{n,gunt}(i) := < i, n >$.
2) Abreviando la notación, la aplicación $h_{B_{imp_{\mathtt{GG}}}}\colon \mathbb{N} \to F^{\mathbb{Z}\times\mathbb{Z}}_{imp_{\mathtt{GG}}}$ viene dada, para cada $n \in \mathbb{N}$, por $h_{B_{imp_{\mathtt{GG}}}}(n) := (< i, x_1 + x_2 - i >, < i, 2*i - x >, < i, i >)$. Observar que la imagen del morfismo final es, en este caso, una única tupla de funciones.
3) $h_{C_{imp_{\mathtt{GG}}}}\colon \mathbb{N} \to F^{\mathbb{Z}\times\mathbb{Z}}_{imp_{\mathtt{GG}}}$ viene definida por $h_{C_{imp_{\mathtt{GG}}}}(n) := (< i, x_1 + x_2 - (n+i) >, < i, 2*(n+i) - x >, < i, n+i >)$, para cada $n \in \mathbb{N}$.

Vamos a dar otros ejemplos de $\mathtt{GG}_{imp}$-álgebras fijando un soporte diferente para el género *elm*. Tomamos el conjunto:

$$D = \bigcup_{n\in\mathbb{Z}} \left(\{n\} \times \mathbb{Z}^n\right)$$

donde, como es habitual, $\mathbb{Z}^n$ denota al producto cartesiano $\mathbb{Z}\times \overset{(n}{\ldots} \times\mathbb{Z}$. Por convenio, si $n \leq 0$, $\mathbb{Z}^n$ es un conjunto unipuntual que denotaremos por $\{*_n\}$. Notar que, en principio, este soporte no es de la forma $\mathbb{Z} \times -$. Una vez fijado este soporte para el género *elm*, tomamos como función *grado* la función $grado(< n, x >) := n$, para todo $n \in \mathbb{Z}$ y para todo $x \in \mathbb{Z}^n$. Denotamos por $\tilde{\mathcal{C}}^D(\mathtt{GG}_{imp})$ a la subcategoría plena de $\tilde{\mathcal{C}}(\mathtt{GG}_{imp})$ con objetos aquellos que tienen al conjunto D anterior como soporte para *elm*. Damos ahora tres ejemplos de álgebras de $\tilde{\mathcal{C}}^D(\mathtt{GG}_{imp})$. En todos los casos el soporte para el género $imp_{\mathtt{GG}}$ será $\mathbb{N}$.

1) Consideramos la $\mathtt{GG}_{imp}$-álgebra A con operaciones definidas en cada componente de modo análogo a las del álgebra A de $\tilde{\mathcal{C}}^{\mathbb{Z}\times\mathbb{Z}}(\mathtt{GG}_{imp})$ (definida en la página 97), extendiendo en cada grado la operación de $\mathbb{Z}$ a $\mathbb{Z}^n$.

   Así tenemos un álgebra de $\tilde{\mathcal{C}}^D(\mathtt{GG}_{imp})$ que representa a la familia de grupos graduados $\{A_n\}_{n\in\mathbb{N}}$ donde, para cada $n \in \mathbb{N}$, A_n es un grupo graduado isomorfo al grupo graduado $\{A_{n,i}\}_{i\in\mathbb{Z}}$ siendo, $A_{n,i} = \{*_i\}$, para cada $i \leq 0$, y $A_{n,i} = (\mathbb{Z}^{\rightarrow n})^n$, para cada $i > 0$.

2) Análogamente definimos una $\mathtt{GG}_{imp}$-álgebra B basada en la segunda de las álgebras de $\tilde{\mathcal{C}}^{\mathbb{Z}\times\mathbb{Z}}(\mathtt{GG}_{imp})$ presentadas anteriormente (en la página 97).

   Esta $\mathtt{GG}_{imp}$-álgebra representa a la familia de grupos graduados indexada por los naturales en la que cada elemento de la familia es isomorfo al grupo graduado $B_n = \{B_{n,i}\}_{i\in\mathbb{Z}}$ siendo $B_{n,i} = (\mathbb{Z}^{\rightarrow i})^n$ si $i > 0$ y, en otro caso, $B_{n,i}$ es el grupo trivial. Como puede verse, la familia está formada por un único grupo graduado.

3) De modo análogo a la definición de la $\mathtt{GG}_{imp}$-álgebra C (definida en la página 97), se define el álgebra que representa a la familia de grupos graduados indexada por los naturales, en la que cada elemento de la familia es isomorfo al grupo graduado $C_n = \{C_{n,i}\}_{i\in\mathbb{Z}}$ siendo $C_{n,i} = (\mathbb{Z}^{\rightarrow n+i})^n$ si $i > 0$ y $C_{n,i}$ el grupo trivial, en otro caso.

La categoría $\tilde{\mathcal{C}}^{D,\{\}}(\mathtt{GG}_{imp})$ tendrá su correspondiente objeto final, que admitirá una descripción en términos de tuplas funcionales análoga a la vista en el primer grupo de ejemplos.

En las dos categorías de $\mathtt{GG}_{imp}$-álgebras que hemos considerado (fijando distintos soportes), hemos tomado para el género *elm* un soporte definido por una unión disjunta, de forma que cada elemento contiene una componente específica que indica el grado al que pertenece. En el soporte del último ejemplo hay información redundante, en el sentido de que la información que aporta la primera componente de cada elemento, realmente está implícita en la segunda y, por tanto, no es necesaria. Esto corresponde formalmente a tomar como soporte para *elm* el conjunto $\bar{D} = \mathbb{Z}^*$ donde $\mathbb{Z}^*$ denota a la unión $\bigcup_{n\in\mathbb{Z}} \mathbb{Z}^n$ siendo, para cada $n < 1$, $\mathbb{Z}^n = \{*_n\}$.

Evidentemente, D y $\bar{D}$ son conjuntos biyectivos, lo que da como consecuencia que las categorías $\tilde{\mathcal{C}}^D(\mathtt{GG}_{imp})$ y $\tilde{\mathcal{C}}^{\bar{D}}(\mathtt{GG}_{imp})$ sean isomorfas.

Así, podemos trabajar tomando como soporte para el género distinguido de la signatura de partida, distintas "representaciones" de la misma reunión disjunta, lo que nos permite elegir en cada caso la que nos resulte más cómoda, ya sea teóricamente o en la máquina.

El hecho de tener que fijar el soporte para el género *elm* parece, en principio, una condición muy rígida, ya que se fija un conjunto en el que están los elementos de todos los grupos de los grupos graduados representados. Entonces la cuestión que se plantea es ¿qué ocurre si se pretenden representar familias de grupos graduados con soportes diferentes pero además diferentes también en cada grado? Por ejemplo, ¿cómo representar la familia de grupos graduados $\{G_n\}_{n\in\mathbb{N}}$ siendo, para cada $n \in \mathbb{N}$, G_n el grupo graduado $G_n = \{\mathbb{Z}/(i+n)\mathbb{Z}\}_{i\in\mathbb{Z}}$? En la familia anterior el conjunto subyacente varía para cada grado y para cada $n \in \mathbb{N}$, luego, es un caso que no queda recogido con lo desarrollado hasta ahora. En general, un grupo $\mathbb{Z}/m\mathbb{Z}$ siempre se puede "representar" sobre $\mathbb{Z}$, incluso sin pasar al cociente. Hay, por lo menos, dos formas de hacerlo:

1) Considerar como soporte todos los elementos de $\mathbb{Z}$ sin pasar al cociente, pero cambiando la *igualdad*, es decir, identificar en cada caso aquellos elementos que interese.

2) Fijar un subconjunto de representantes canónicos (como hemos visto en el ejemplo 2.2.2).

La primera de las dos opciones tiene que ver con la representación y con la implementación, ya en el paso a la máquina, y la trataremos en el siguiente capítulo. La segunda opción está relacionada con la parcialidad. Como hemos visto, llevaría a fijar los dominios de definición de los operadores de la signatura. Haría falta una parcialidad entendida en un sentido más amplio que lo estudiado en este capítulo. Para recoger este caso, será necesario que los dominios de los operadores puedan ser diferentes entre las distintas álgebras de la familia. Esto se abordará en el cuarto capítulo.

Por último, vamos a ver un tipo de signatura para grupos graduados más próxima a la usada en EAT. En este sentido, tomamos una signatura que contiene explícitamente la operación *grado*, en lugar de exigir su existencia como operador dentro de las álgebras, como ocurría en `GG` (hasta aquí, el *grado* no se ha considerado como una operación del grupo graduado, sino como un operador que debían poseer las álgebras). Partimos de la signatura `GG` y añadimos *grado* como operación, así obtenemos la siguiente signatura que denotamos por $\overline{\texttt{GG}}$

$$\begin{array}{lcccl} gprd & : & elm \;\; elm & \to & elm \\ ginv & : & elm & \to & elm \\ gunt & : & int & \to & elm \\ grado & : & elm & \to & int \end{array}$$

Consideramos ahora una subcategoría de $\overline{\texttt{GG}}$-álgebras que denotamos por $\mathcal{C}(\overline{\texttt{GG}})$, formada por las álgebras A que verifican las siguientes condiciones:

i) El soporte para el género *int* es $\mathbb{Z}$.

ii) El dominio de definición de la función $gprd_A$ es el siguiente:

$$Def(gprd_A) = \left\{(x,y) \in A_{elm} \times A_{elm} \;\mid\; grado_A(x) = grado_A(y)\right\}$$

iii) Las funciones $ginv_A$, $gunt_A$ y $grado_A$ son totales.

iv) La función $grado_A$ tiene que ser compatible con las operaciones, es decir, debe verificarse lo siguiente:

- para cada $n \in \mathbb{Z}$, $grado_A(gunt_A(n)) = n$;
- para cada $x \in A_{elm}$ y cada $n \in \mathbb{Z}$ tales que $grado_A(x) = n$, se tiene que $grado_A(ginv_A(x)) = n$;
- para cada par $x, y \in A_{elm}$ y cada $n \in \mathbb{Z}$ tales que $grado_A(x) = n$ y $grado_A(y) = n$, se tiene que $grado_A(gprd_A(x, y)) = n$.

v) Para cada $n \in \mathbb{Z}$, el conjunto $C_n = \{x \in A_{elm} \mid grado_A(x) = n\}$, junto con las operaciones $gprd_A|_{C_n \times C_n}$, $ginv_A|_{C_n}$ y $gunt_A|_{\{n\}}$ tiene estructura de grupo.

Como morfismos tomamos únicamente aquellos morfismos de $\overline{\mathtt{GG}}$-álgebras que son la identidad sobre $\mathbb{Z}$.

Análogamente a lo visto para $\tilde{\mathcal{C}}(\mathtt{GG})$, es claro que cada álgebra de $\mathcal{C}(\overline{\mathtt{GG}})$ determina un grupo graduado y viceversa, dando lugar a otra categoría con el mismo poder expresivo que la de grupos graduados. Consideramos el TAD $\mathcal{T}_{\overline{\mathtt{GG}}} = \langle \overline{\mathtt{GG}}, \mathcal{C}(\overline{\mathtt{GG}}) \rangle$ al que le aplicamos la operación $()_{imp}$, es decir, consideramos la signatura $\overline{\mathtt{GG}}_{imp}$ cuyas operaciones son las siguientes:

$$\begin{array}{rcccccc}
imp_gprd & : & imp_{\mathtt{GG}} & elm & elm & \rightarrow & elm \\
imp_ginv & : & imp_{\mathtt{GG}} & elm & & \rightarrow & elm \\
imp_gunt & : & imp_{\mathtt{GG}} & int & & \rightarrow & elm \\
imp_grado & : & imp_{\mathtt{GG}} & elm & & \rightarrow & int
\end{array}$$

y nos restringimos a la correspondiente subcategoría de $PAlg(\overline{\mathtt{GG}}_{imp})$, que denotamos por $\mathcal{C}(\overline{\mathtt{GG}}_{imp})$ cuyos objetos son las $\overline{\mathtt{GG}}_{imp}$-álgebras tales que, cada elemento del soporte del género $imp_{\mathtt{GG}}$ define un álgebra de $\mathcal{C}(\overline{\mathtt{GG}})$ y con morfismos identidad sobre $\mathbb{Z}$. Así queda definido el TAD $\mathcal{T}_{\overline{\mathtt{GG}}_{imp}} = \langle \overline{\mathtt{GG}}_{imp}, \mathcal{C}(\overline{\mathtt{GG}}_{imp}) \rangle$.

El hecho de que las interpretaciones de la operación *grado* deban ser totales, hacen que, de nuevo, el soporte para *elm* pueda siempre ser considerado como una representación de la unión disjunta de las preimágenes de la función *grado*. Por ello, podemos siempre tomar como soporte para *elm* un conjunto de la forma $\mathbb{Z} \times D$ y con ello, dar por fijada la interpretación de la operación *imp_grado* (proyección sobre la primera componente del elemento). Si A es un álgebra de $\mathcal{C}(\overline{\mathtt{GG}}_{imp})$ con soporte $\mathbb{Z} \times D$ para *elm*, entonces $imp\_grado_A \colon A_{imp_{\mathtt{GG}}} \times (\mathbb{Z} \times D) \rightarrow \mathbb{Z}$ se define por $imp\_grado_A(a, < i, d >) := i$, para algunos $a \in A_{imp_{\mathtt{GG}}}$, para todo $i \in \mathbb{Z}$ y para todo $d \in D$. Como puede verse en este caso, la definición de la operación imp_grado_A no depende del primer argumento, es decir, podemos verla como una función $imp\_grado_A \colon (\mathbb{Z} \times D) \rightarrow \mathbb{Z}$, función cuya definición es fija una vez determinado el soporte para *elm*. Por tanto, estamos en la misma situación que en el caso de no incluir la operación *grado* en la signatura. Luego, las dos signaturas vistas para los grupos graduados, $\mathtt{GG}$ y $\overline{\mathtt{GG}}$, nos conducen en la práctica a situaciones equivalentes (el grado puede ser o no una componente explícita del elemento).

Desde el punto de vista de las Especificaciones Ocultas, la interpretación de lo anterior es que la operación *imp_grado* es una operación visible, no depende del género $imp_{\mathtt{GG}}$. Esto nos

lleva a ver una segunda razón por la que podemos encontrarnos operaciones visibles en una signatura del tipo *imp*: operaciones que no dependan del primer argumento.

Antes de finalizar el ejemplo, señalar que, además de tratar en él el tema de la graduación, ha dado lugar a que aparezcan otras cuestiones que creemos merece la pena destacar:

- Hemos visto dos razones por las que en una signatura pueden aparecer operaciones visibles:
  - Una es la necesidad de generar los elementos del álgebra de datos, ya que lo natural es construirla a partir de una signatura, tomando como álgebra de datos el álgebra inicial.
  - La otra es la aparición en la signatura de operaciones que, al fijar los soportes para los géneros de la signatura de partida, no dependen del género distinguido. En ese caso su interpretación está determinada por la elección de los soportes y es fija para todas las álgebras con esos soportes.
- A lo largo de todo el capítulo, hemos trabajado con una aproximación semántica basada en modelos, es decir, hemos trabajado con signaturas y la forma de restringir la categoría de álgebras ha sido imponiendo condiciones a éstas, de modo que no hemos utilizado explícitamente axiomas. Sin embargo, hemos mostrado que en el caso tratado en el ejemplo, es indistinto hacerlo así o añadiendo axiomas a la signatura.
- Hasta donde hemos llegado con el ejemplo no cubrimos todos los casos. Como ya hemos comentado, será necesario enriquecer lo que entendemos por "representación" de las estructuras para poder cubrirlos. Esto tiene que ver con el paso de las álgebras a la máquina y de ello nos ocuparemos en los próximos capítulos.

2.9.2 Conjuntos Simpliciales

Un *conjunto simplicial* K es un conjunto graduado indexado por los enteros no negativos, $K = \{K^n\}_{n\in\mathbb{N}}$, junto con dos familias de funciones

- $\delta_i^n : K^n \to K^{n-1}$, para $n > 0$ e $i = 0, \ldots, n$
- $\eta_i^n : K^n \to K^{n+1}$, para $n \geq 0$ e $i = 0, \ldots, n$

que satisfacen las siguientes identidades:

$$\begin{aligned}
\delta_i^{n-1}\delta_j^n &= \delta_{j-1}^{n-1}\delta_i^n \quad \text{si} \quad i < j,\\
\eta_i^{n+1}\eta_j^n &= \eta_{j+1}^{n+1}\eta_i^n \quad \text{si} \quad i \leq j,\\
\delta_i^{n+1}\eta_j^n &= \eta_{j-1}^{n-1}\delta_i^n \quad \text{si} \quad i < j,\\
\delta_j^{n+1}\eta_j^n &= \text{identidad} = \delta_{j+1}^{n+1}\eta_j^n,\\
\delta_i^{n+1}\eta_j^n &= \eta_j^{n-1}\delta_{i-1}^n \quad \text{si} \quad i > j+1.
\end{aligned}$$

A los elementos de K^n se les denomina *símplices de dimensión* n o n*-símplices* de K. A las funciones δ_i^j se les denomina *operadores cara* y a las η_i^j *operadores de degeneración*. Un símplice que se encuentra en la imagen de un operador de degeneración se denomina *símplice degenerado* y, en otro caso, *símplice geométrico* o *no degenerado*. Obviamente, esta nomenclatura indica el origen geométrico (topológico) de todas estas nociones.

Si K y L son conjuntos simpliciales, un morfismo $f\colon K \to L$ entre ellos es una familia de aplicaciones, $\{f^n\colon K^n \to L^n\}_{n\in\mathbb{N}}$, entre los conjuntos correspondientes al mismo grado que conmutan con los operadores cara y degeneración, es decir, verifican

- $f^{n-1}\delta_i = \delta_i f^n$, para todo $n > 0$ y todo $i = 0, \ldots, n$.
- $f^{n+1}\eta_i = \eta_i f^n$, para todo $n \geq 0$ y todo $i = 0, \ldots, n$.

Los libros [78] y [73] son referencias en las que se trabaja con objetos simpliciales y en las que pueden encontrarse las definiciones anteriores sobre conjuntos simpliciales, así como un estudio más detallado de éstos.

Los conjuntos simpliciales junto con los morfismos entre ellos forman una categoría que denominamos Categoría de Conjuntos Simpliciales y que denotamos por *CSimp*. La estructura conjunto simplicial es una de las fundamentales en Topología Algebraica ya que proporciona modelos combinatorios (discretos) de espacios topológicos y de objetos geométricos. Por ello, es útil en todo lo que tiene que ver con cálculos de invariantes, en particular, también es fundamental a la hora de simular en la máquina el trabajo con espacios topológicos.

Por ejemplo, un tetraedro puede venir representado por el siguiente conjunto simplicial:

- en dimensión 0, los vértices $(v0), (v1), (v2), (v3)$ son 0-símplices;
- en dimensión 1, las aristas: $(v0, v1), (v0, v2), (v0, v3), (v1, v2), (v1, v3), (v2, v3)$;
- en dimensión 2, las caras: $(v0, v1, v2), (v0, v1, v3), (v0, v2, v3), (v1, v2, v3)$;
- en dimensión 3, el propio tetraedro $(v0, v1, v2, v3)$.

Los objetos anteriores, que tienen interpretación geométrica, serán los símplices geométricos. A partir de los objetos anteriores, para reconstruir totalmente el tetraedro hace falta establecer cómo esos símplices están relacionados entre sí. Es ahí donde aparecen los operadores cara y degeneración. En este caso concreto, un operador δ_i^n actua eliminando el vértice que ocupa la posición $i-1$ en la lista sobre la que trabaja, mientras que cada operador η_i^n duplica el vértice que ocupa la posición $i-1$. Notar que los símplices obtenidos por aplicación de operadores de degeneración no tienen significado geométrico. En el ejemplo se ve bien la idea geométrica intuitiva de la noción de cara: un símplice de dimensión n tiene $n+1$ caras, cada una de las cuales se obtiene eliminando un vértice.

Nuestro objetivo es modelar la categoría *CSimp* y para ello, vamos a definir un TAD que la aproxime. En [66] publicamos un estudio de este caso aunque con menos detalle. Así, se trata de obtener una representación para los conjuntos simpliciales a partir de una signatura y una adecuada categoría de álgebras. Teniendo en cuenta que un conjunto simplicial tiene como base un conjunto graduado, podemos pensar en una signatura del tipo de la que hemos considerado para los grupos graduados, es decir, una signatura que posea un género con el que representar todos los símplices del conjunto simplicial (todos los símplices van a vivir

juntos), y otro género asociado a la graduación. A estos géneros los denotaremos *smp* y *nat*, respectivamente. Un conjunto simplicial no posee operaciones propiamente dichas, sin embargo, lo que sí hay son operaciones entre los conjuntos correspondientes a grados consecutivos. Una forma de representar lo anterior, tal y como se ha hecho en el ejemplo de los grupos graduados, consiste en considerar las dos operaciones siguientes:

- operación que calcula la cara i-ésima de un símplice; tiene como argumentos: el índice de la cara a calcular, la dimensión del símplice y el símplice en sí;
- operación que calcula la i-ésima degeneración de un símplice; tiene como argumentos: el índice de la degeneración, la dimensión del símplice y el símplice en sí.

Gráficamente, la idea consiste en disponer de un único δ y un único η, pasando a ser el subíndice y el superíndice argumentos de estos operadores. Con todo esto, llamamos `CS` a la siguiente signatura, con la que pretendemos "representar" conjuntos simpliciales:

$$\delta \quad : \quad nat \quad nat \quad smp \quad \to \quad smp$$

$$\eta \quad : \quad nat \quad nat \quad smp \quad \to \quad smp$$

Obviamente, no toda `CS`-álgebra representa a un conjunto simplicial. Para definir nuestro TAD de los conjuntos simpliciales nos restringimos a una subcategoría de `CS`-álgebras. Llamamos $\mathcal{C}(\texttt{CS})$ a la subcategoría de $PAlg(\texttt{CS})$ cuyos objetos son las álgebras A que verifican las siguientes condiciones:

i) El conjunto soporte para el género *nat* es $\mathbb{N}$.

ii) Existe una función total $dim_A \colon A_{smp} \to \mathbb{N}$, que asocia a cada elemento de A_{smp} una dimensión.

iii) $Def(\delta_A) = \{(i,n,x) \in \mathbb{N} \times \mathbb{N} \times A_{smp} \mid 0 \leq i \leq n,\ n > 0,\ dim_A(x) = n\}$

iv) $Def(\eta_A) = \{(i,n,x) \in \mathbb{N} \times \mathbb{N} \times A_{smp} \mid 0 \leq i \leq n,\ n \geq 0,\ dim_A(x) = n\}$

v) Las operaciones tienen que ser compatibles con la función dim_A. Es decir, debe verificarse lo siguiente:

- toda terna $(i,n,x) \in Def(\delta_A)$ con $dim_A(x) = n$ cumple que $dim_A(\delta_A(i,n,x)) = n-1$;
- toda terna $(i,n,x) \in Def(\eta_A)$ con $dim_A(x) = n$ cumple que $dim_A(\eta_A(i,n,x)) = n+1$.

vi) La `CS`-álgebra A verifica las identidades exigidas a los conjuntos simpliciales, identidades que pueden describirse del siguiente modo:

- para cada $x \in A_{smp}$ con $dim_A(x) = n$ y cada par $i,j \in \mathbb{N}$ tal que $i,j < n$ e $i < j$, se tiene que $\delta_A(i,n-1,\delta_A(j,n,x)) = \delta_A(j-1,n-1,\delta_A(i,n,x))$;
- para cada $x \in A_{smp}$ con $dim_A(x) = n$ y cada par $i,j \in \mathbb{N}$ tal que $i,j \leq n$ e $i \leq j$, se tiene que $\eta_A(i,n+1,\eta_A(j,n,x)) = \eta_A(j+1,n+1,\eta_A(i,n,x))$;

- para cada $x \in A_{smp}$ con $dim_A(x) = n$ y cada par $i, j \in \mathbb{N}$ tales que $i, j \leq n$ e $i < j$, se tiene que $\delta_A(i, n+1, \eta_A(j, n, x)) = \eta_A(j-1, n-1, \delta_A(i, n, x))$;
- para cada $x \in A_{smp}$ con $dim_A(x) = n$ y cada $j \in \mathbb{N}$ con $j \leq n$, se tiene que $\delta_A(j, n+1, \eta_A(j, n, x)) = \delta_A(j+1, n+1, \eta_A(j, n, x)) = x$;
- para cada $x \in A_{smp}$ con $dim_A(x) = n$ y cada par $i, j \in \mathbb{N}$ tal que $i \leq n+1$ e $i > j+1$, se tiene que $\delta_A(i, n+1, \eta_A(j, n, x)) = \eta_A(j, n-1, \delta_A(i-1, n, x))$.

Como morfismos en $\mathcal{C}(\mathtt{CS})$ tomamos aquellos morfismos de $\mathtt{CS}$-álgebras que son la identidad sobre $\mathbb{N}$. Observar que si $f\colon A \to B$ es un morfismo entre álgebras de $\mathcal{C}(\mathtt{CS})$ que es la identidad sobre el género *nat*, entonces f respeta las dimensiones, es decir, si $a \in A_{smp}$ con $dim_A(a) = n$, entonces $dim_B(f_{smp}(a)) = n$.

Funtores similares a los definidos para grupos graduados inducen una equivalencia entre las categorías $\mathcal{C}(\mathtt{CS})$ y *CSimp*. En este ejemplo, la función *dim* juega el mismo papel que la función *grado* en el caso de los grupos graduados y su existencia es la que permite establecer dicha equivalencia.

Consideramos $\mathcal{T}_{\mathtt{CS}} = \langle \mathtt{CS}, \mathcal{C}(\mathtt{CS}) \rangle$, al que denominamos *TAD de los Conjuntos Simpliciales.*

Aplicándole la operación $()_{imp}$ obtenemos el TAD $\mathcal{T}_{\mathtt{CS}_{imp}} = \langle \mathtt{CS}_{imp}, \mathcal{C}(\mathtt{CS}_{imp}) \rangle$, donde $\mathtt{CS}_{imp}$ es la signatura con géneros *nat*, *smp* e $imp_{\mathtt{CS}}$ y con operaciones

$$imp\_\delta \quad : \quad imp_{\mathtt{CS}} \quad nat \quad nat \quad smp \quad \to \quad smp$$

$$imp\_\eta \quad : \quad imp_{\mathtt{CS}} \quad nat \quad nat \quad smp \quad \to \quad smp$$

$\mathcal{C}(\mathtt{CS}_{imp})$ es la subcategoría plena de $PAlg(\mathtt{CS}_{imp})$ con objetos las $\mathtt{CS}_{imp}$-álgebras $A = \langle A_{imp_{\mathtt{CS}}}, A_{smp}, \mathbb{N}, \{(imp\_\delta_A,\ Def(imp\_\delta_A)),\ (imp\_\eta_A,\ Def(imp\_\eta_A))\} \rangle$, tales que, para todo $a \in A_{imp_{\mathtt{CS}}}$, la $\mathtt{CS}$-álgebra $A_a = \langle A_{smp}, \mathbb{N}, \{imp\_\delta_A(a,-,-,-),\ Def(imp\_\delta_A(a,-,-,-))), (imp\_\eta_A(a,-,-,-),\ Def(imp\_\delta_A(a,-,-,-)))\} \rangle$ está en $\mathcal{C}(\mathtt{CS})$.

Al igual que en lo anterior, cada objeto de la categoría $\mathcal{C}(\mathtt{CS}_{imp})$ será visto como una familia indexada de conjuntos simpliciales.

Se dan a continuación algunos ejemplos de $\mathtt{CS}_{imp}$-álgebras. El primero de ellos va a modelar a una familia de complejos simpliciales. Un *complejo simplicial* K es un subconjunto de las partes finitas de un conjunto dado, de forma que K es cerrado por la formación de subconjuntos no vacíos. Es decir, si un subconjunto x pertenece a un complejo simplicial K, todos los subconjuntos no vacíos de x también pertenecen a K. Así, un complejo simplicial se construye a partir de un conjunto dado, a cuyos elementos se les denomina vértices. Si en el conjunto de vértices hay definida una relación de orden, entonces hablaremos de *complejo simplicial ordenado.*

A partir de un complejo simplicial ordenado se define de modo natural un conjunto simplicial. La idea consiste en tomar como n-símplices las listas ordenadas de forma no decreciente con exactamente $n+1$ vértices. En este caso, todos los subconjuntos de un n-símplice dan lugar a símplices de la dimensión correspondiente. Sin entrar en detalles, los operadores cara eliminan un elemento y los de degeneración lo añaden (duplicándolo). En realidad, una forma adecuada de representar cada símplice de un complejo simplicial ordenado es mediante una lista ordenada de vértices.

Un ejemplo de complejo simplicial ordenado es el símplice estándar de dimensión n, denotado en la literatura por Δ^n. Sus vértices son los naturales entre 0 y n. Un símplice en una dimensión k, con $k \geq 0$, es una lista ordenada con exactamente $k+1$ naturales entre 0 y n. El operador degeneración i-ésimo duplica el elemento que ocupa la posición $i+1$ en la lista, sea en la dimensión que sea. El operador cara i-ésima elimina el elemento que ocupa la posición $i+1$. Notar que la diferencia entre símplices degenerados y símplices geométricos está en contener o no elementos repetidos. Así, el tetraedro definido anteriormente es la versión conjunto simplicial de Δ^3.

Se puede representar la familia de complejos simpliciales $\{\Delta^m\}_{m\in\mathbb{N}}$ por medio de la $\mathtt{CS}_{imp}$-álgebra Δ que definimos a continuación. Tomamos como soporte para el género *smp* el conjunto Δ_{smp} formado por las listas no decrecientes y no vacías de enteros no negativos y el conjunto $\mathbb{N}$ como soporte en Δ para el género $imp_{\mathtt{CS}}$. Las operaciones vienen dadas por:

- $imp\_\delta_\Delta \colon \mathbb{N}\times\mathbb{N}\times\mathbb{N}\times\Delta_{smp} \to \Delta_{smp}$ función parcial con dominio $Def(imp\_\delta_\Delta) = \{(m,i,n,(j_1,\ldots,j_{n+1}) \in \mathbb{N}\times\mathbb{N}\times\mathbb{N}\times\Delta_{smp} \mid i \leq n,\ n>0 \text{ y } 0\leq j_1\leq\cdots\leq j_{n+1}\leq m\}$ definida por $imp\_\delta_\Delta(m,i,n,(j_1,\ldots j_{n+1})) := (j_1,\ldots,j_i,j_{i+2},\ldots,j_{n+1})$, para cada $(m,i,n,(j_1,\ldots j_{n+1})) \in Def(imp\_\delta_\Delta)$;

- $imp\_\eta_\Delta \colon \mathbb{N}\times\mathbb{N}\times\mathbb{N}\times\Delta_{smp} \to \Delta_{smp}$ función parcial con dominio $Def(imp\_\eta_\Delta) = \{(m,i,n,(j_1,\ldots,j_{n+1}) \in \mathbb{N}\times\mathbb{N}\times\mathbb{N}\times\Delta_{smp} \mid i\leq n,\ n\geq 0 \text{ y } 0\leq j_1\leq\cdots\leq j_{n+1}\leq m\}$ definida por $imp\_\eta_\Delta(m,i,n,(j_1,\ldots j_{n+1})) := (j_1,\ldots,j_i,j_{i+1},j_{i+1},\ldots,j_{n+1})$, para cada $(m,i,n,(j_1,\ldots j_{n+1})) \in Def(imp\_\eta_\Delta)$.

Notar que la $\mathtt{CS}_{imp}$-álgebra Δ no es un objeto de la categoría $\mathcal{C}(\mathtt{CS}_{imp})$. En efecto, dado un $m\in\mathbb{N}$, la correspondiente $\mathtt{CS}$-álgebra Δ_m que se obtiene al fijar m en cada operación, no está en $\mathcal{C}(\mathtt{CS})$. Notar que los dominios de definición de las operaciones de Δ_m no satisfacen las condiciones para estar en $\mathcal{C}(\mathtt{CS})$, ya que aparece una dependencia del primer argumento (de m). El motivo es que estamos intentando representar una familia de conjuntos simpliciales con soportes diferentes pero además distintos en cada dimensión.

Vamos a ver qué ocurre con otra representación, en principio bastante diferente, de la familia $\{\Delta^m\}_{m\in\mathbb{N}}$. Consideramos el complejo simplicial ordenado con vértices $\mathbb{N}$, que denotamos por Δ^∞ (todo multiconjunto finito es un símplice). Podemos representar Δ^∞ como la $\mathtt{CS}$-álgebra que tiene como soporte para *smp* el conjunto de listas no decrecientes (y no vacías) de naturales (siendo los símplices no degenerados aquellas listas estrictamente crecientes) y de forma que una lista de longitud $l+1$ es un símplice de dimensión l, para cualquier $l\in\mathbb{N}$. Los operadores degeneración y cara son la extensión natural de los de Δ^k, con k finito. Para cada $m\in\mathbb{N}$, $\Delta^m \subsetneq \Delta^\infty$, así podemos tomar como representación de la familia de conjuntos simpliciales $\{\Delta^m\}_{m\in\mathbb{N}}$ a $\{\Delta^\infty\}_{m\in\mathbb{N}}$, ya que dado un símplice cualquiera de Δ^∞ y dado un natural m, sabemos si es o no símplice de Δ^m. La familia $\{\Delta^m\}_{m\in\mathbb{N}}$ puede describirse como $\mathtt{CS}_{imp}$-álgebra mediante una pequeña variación del álgebra Δ del ejemplo anterior, bastaría con eliminar las condiciones sobre el argumento m en los dominios de definición de los operadores. Parece que con esto hemos solucionado el problema que tenía Δ. Sin embargo, esto no es así, para que fuera realmente una representación de la familia anterior, deberíamos poder considerar solamente los símplices que interesan en cada caso, pero teniendo en cuenta que el dominio de definición está determinado, esto no es posible.

Lo anterior parece conducirnos a que la construcción que tenemos es demasiado rígida, solo permite considerar $\mathtt{CS}_{imp}$-álgebras en las que los dominios de definición de sus operaciones

factoricen en la componente asociada al género que se utiliza para indexar a las álgebras de la familia. Sin embargo, como acabamos de ver, existen representaciones muy naturales que, de esta forma, no quedan recogidas.

Otro tipo de ejemplos de familias de conjuntos simpliciales que nos interesan, de momento a nivel teórico, son los objetos finales de determinadas subcategorías de $\mathtt{CS}_{imp}$-álgebras, obtenidas fijando los conjuntos soporte para los géneros *nat* y *smp*, tal y como hemos visto en el desarrollo del capítulo.

Dado un conjunto graduado $K = \{K^n\}_{n\in\mathbb{N}}$, vamos a definir la "$\mathtt{CS}_{imp}$-álgebra de todos los conjuntos simpliciales construidos sobre K". Fijamos $\mathbb{N}$ como soporte para el género *nat*, K como soporte para *smp* y definimos la $\mathtt{CS}_{imp}$-álgebra $\mathbb{1}^K$ como sigue

- $\mathbb{1}^K_{imp_{\mathtt{CS}}} = \big\{(d,g) \in K^{\mathbb{N}\times\mathbb{N}\times K} \times K^{\mathbb{N}\times\mathbb{N}\times K} \mid \langle K, \mathbb{N}, \{d,g\}\rangle \text{ es conjunto simplicial}\big\}$; es decir, $\langle K, \mathbb{N}, \{d,g\}\rangle$ es un objeto de la categoría $\mathcal{C}(\mathtt{CS})$.
- $imp\_\delta_{\mathbb{1}^K}\colon \mathbb{1}^K_{imp_{\mathtt{CS}}} \times \mathbb{N} \times \mathbb{N} \times K \to K$ función parcial con dominio $Def(imp\_\delta_{\mathbb{1}^K}) = \big\{\big((d,g),i,n,x\big) \in \mathbb{1}^K_{imp_{\mathtt{CS}}} \times \mathbb{N} \times \mathbb{N} \times K \mid (i,n,x) \in Def(d)\big\}$ y definida, para cada $\big((d,g),i,n,x\big) \in Def(imp\_\delta_{\mathbb{1}^K})$ como $imp\_\delta_{\mathbb{1}^K}\big((d,g),i,n,x\big) := d(i,n,x)$.
- Análogamente se define $imp\_\eta_{\mathbb{1}^K}$.

Es claro que $\mathbb{1}^K$ es un objeto de la categoría $\mathcal{C}(\mathtt{CS}_{imp})$. Intuitivamente, la $\mathtt{CS}_{imp}$-álgebra $\mathbb{1}^K$ representa a la familia formada por todos los conjuntos simpliciales que pueden contruirse sobre el conjunto graduado K. Formalmente, la propiedad anterior queda recogida en el siguiente teorema (que ya incluimos en [66]), que se obtiene aplicando el teorema general 2.4.6, al caso de los conjuntos simpliciales construidos sobre un conjunto graduado:

Teorema 2.9.1 *La $\mathtt{CS}_{imp}$-álgebra $\mathbb{1}^K$ es objeto final en la categoría con objetos las familias de conjuntos simpliciales que pueden definirse sobre un conjunto graduado K, y con morfismos aquellos que son la identidad sobre $\mathbb{N}$ y sobre K.*

Al trabajar con conjuntos simpliciales, es habitual partir de un conjunto graduado $K = \{K^n\}_{n\in\mathbb{N}}$ que contenga, no a todos los símplices (tal y como hemos hecho hasta ahora), sino solamente a los símplices geométricos.

Sea $K = \{K^n\}_{n\in\mathbb{N}}$ un conjunto graduado fijo. Nuestro objetivo es representar familias de conjuntos simpliciales cuyos símplices geométricos sean los de K. Para ello, comenzamos definiendo el conjunto soporte para el género *smp*. Lo natural en este caso es pensar en K como conjunto de generadores de símplices. Así, el soporte del género *smp* va a ser definido como el soporte del álgebra inicial para una signatura concreta, utilizando la siguiente propiedad de los conjuntos simpliciales:

Propiedad 2.9.2 Cualquier símplice de un conjunto simplicial puede expresarse de forma única como una sucesión de degeneraciones de un símplice no degenerado; es decir, cada n-símplice x admite una expresión de la forma $x = \eta^{n-1}_{j_k} \dots \eta^{n-k}_{j_1} z$ donde $k \geq 0$, $j_1 < \dots < j_k$ y z es un $(n-k)$-símplice no degenerado. Si $k = 0$ entonces x es un n-símplice no degenerado.

Llamaremos *símplice abstracto* a la representación de un símplice mediante un par formado por una secuencia de degeneraciones aplicables consecutivamente y por un símplice no degenerado. La propiedad anterior nos dice que todo símplice posee representación como símplice

abstracto. En la práctica, para representar un símplice abstracto basta tomar un par formado por un símplice geométrico y una lista, que contiene los índices de las degeneraciones. Apoyándonos en esto, consideramos la signatura `ABSMP`, que tiene géneros *nat*, *gsmp* y *absmp* y operaciones:

$$\begin{array}{lcl} coer & : & gsmp \to absmp \\ \eta & : & nat \ \ nat \ \ absmp \to absmp \end{array}$$

La signatura anterior pretende solo establecer condiciones sobre los símplices. La idea es que el género *gsmp* corresponda a los símplices geométricos y *absmp* a las representaciones como símplices abstractos. Por su parte, la operación *coer* se interpretará como una forma de construir la representación como símplice abstracto de cada símplice geométrico. La representación de cada símplice como abstracto está determinada (es un par formado por una lista de degeneraciones y un símplice geométrico). Esto nos permite incluir el operador de degeneración en la signatura para los símplices, así como su comportamiento, puesto que lo conocemos.

Consideramos la categoría cuyos objetos son las `ABSMP`-álgebras A que verifican las siguientes condiciones:

i) El conjunto soporte para el género *nat* es $\mathbb{N}$.

ii) El conjunto soporte para el género *gsmp* es $\bigsqcup_{n\in\mathbb{N}} K^n$.

iii) Existe una función $dim_A \colon A_{absmp} \to \mathbb{N}$, que asocia a cada elemento de A_{absmp} una dimensión.

iv) $Def(\eta_A) = \{(i,n,x) \in \mathbb{N}\times\mathbb{N}\times A_{absmp} \mid 0 \le i \le n,\ n \ge 0,\ dim_A(x) = n\}$

v) La función dim_A es compatible con las operaciones, es decir, verifica:

- $dim_A(coer_A(x)) = n$ para todo $x \in K^n$; (en particular, $coer_A$ es total);
- para cada $(i,n,x) \in Def(\eta_A)$, se tiene que $dim_A(\eta_A(i,n,x)) = n+1$.

vi) Para cada $x \in A_{absmp}$ con $dim_A(x) = n$ y cada par $i,j \in \mathbb{N}$ con $i,j \le n$ e $i \le j$, se tiene que $\eta_A(i,n+1,\eta_A(j,n,x)) = \eta_A(j+1,n+1,\eta_A(i,n,x))$.

Por medio de las condiciones anteriores hemos fijado los soportes para *nat* y *gsmp* y hemos determinado el comportamiento de los operadores η exigiendo que cada álgebra de la categoría verifique todas las propiedades exigidas al operador de degeneración de un conjunto simplicial.

Como morfismos tomamos aquellos morfismos de `ABSMP`-álgebras que son la identidad sobre los soportes correspondientes a los géneros *nat* y *gsmp*.

La categoría anterior posee objeto inicial, que denotamos por I^K. Una descripción de este objeto es la siguiente:

- El soporte para el género *absmp* viene dado por el siguiente conjunto:

$$I^K_{absmp} = \Big\{\langle (j_k,\dots,j_1), a\rangle \mid \exists\ n \in \mathbb{N} \text{ tal que } a \in K^n,\ k \in \mathbb{N}, j_i \in \mathbb{N} \text{ para todo } i = 1,\dots,k, \text{ y } 0 \le j_1 < \cdots < j_k \le n+k-1\Big\}$$

Por la condición *vi*) impuesta a las álgebras de la categoría, no hay ningún problema en tomar la lista que forma parte de los elementos de I^K_{absmp} estrictamente decreciente. En efecto, en el caso de que la lista contenga un par (l,h) con $l \leq h$, éste puede sustituirse por $(h+1,l)$, de forma que se consigue que sea estrictamente decreciente.

- La dimensión de un símplice abstracto viene dada por la función $dim_{I^K}\colon I^K_{absmp} \to \mathbb{N}$ tal que $dim_{I^K}\big(\langle (j_k,\ldots,j_1),a\rangle\big) := n+k$ siendo n tal que $a \in K^n$.
- La función $coer_{I^K}\colon I^K_{gsmp} \to I^K_{absmp}$ se define como $coer_{I^K}(a) := \langle (\,),a\rangle$.
- El operador de degeneración $\eta_{I^K}\colon \mathbb{N}\times\mathbb{N}\times I^K_{absmp} \to I^K_{absmp}$ se define como

$$\eta_{I^K}\big(i,n,\langle (j_k,\ldots,j_1),a\rangle\big) := \langle (j_k+1,\ldots,j_l+1,i,j_{l-1},\ldots,j_1),a\rangle$$

siendo l tal que $j_{l-1} < i \leq j_l$.

Dicho de otro modo, se añade i al final de la lista de degeneraciones, y se aplica *vi*) hasta obtener una lista estrictamente decreciente.

Notar que por definición de la categoría de `ABSMP`-álgebras que hemos considerado, el dominio de la función η_{I^K} está fijo y es $\big\{\big(i,n,\langle (j_k,\ldots,j_1),a\rangle\big) \;\mid\; 0 \leq i \leq n\ ,\ n \geq\ 0\ ,\ k\ +\ dim_{I^K}(\langle (),a\rangle) = n\big\}$.

Es importante volver a señalar que cualquier `CS`-álgebra de $\mathcal{C}(\texttt{CS})$ que tenga como soporte para *smp* el conjunto I^K_{absmp}, ya tiene determinado el comportamiento del operador de degeneración, que siempre será el dado por η_{I^K}.

El conjunto I^K_{absmp} proporciona una forma de representación, un patrón, para los símplices abstractos. Así, para tener `CS`-álgebras que sean conjuntos simpliciales con símplices geométricos los elementos de K, fijaremos el conjunto I^K_{absmp} como soporte asociado al género *absmp*. Damos a continuación algunos ejemplos de `CS`-álgebras definidas de este modo.

Para cada $m \in \mathbb{N}$, consideramos la esfera S^m, espacio topológico con dos símplices geométricos, uno en dimensión 0 y otro en dimensión m. Consideramos el conjunto graduado $K_m = \{K^n_m\}_{n\in\mathbb{N}}$ donde $K^0_m = \{g_0\}$, $K^m_m = \{g_m\}$ y $K^n_m = \emptyset$ si $n \in \mathbb{N} \setminus \{0,m\}$. Denotamos por S^m a la `CS`-álgebra con conjuntos soporte $\mathbb{N}$ e $I^{K_m}_{absmp}$ para los géneros *nat* y *smp*, respectivamente, y con las siguientes operaciones:

- La operación η_{S^m} viene dada por la función $\eta_{I^{K_m}}$.
- La función $\delta_{S^m}\colon \mathbb{N}\times\mathbb{N}\times I^{K_m}_{absmp} \to I^{K_m}_{absmp}$ tiene por dominio $Def(\delta_{S^m}) = \big\{\big(i,n,\langle (j_k,\ldots,j_1),a\rangle\big) \in \mathbb{N}\times\mathbb{N}\times I^K_{absmp} \;\mid\; i \leq n,\ n>0$ y $\exists\, l \in \mathbb{N}$ tal que $a \in K^l_m$ con $k+l=n\big\}$. En este caso concreto, para $(i,n,x) \in Def(\delta_{S^m})$ solo se tienen dos posibilidades para x: $x = \langle (n-1,\ldots,0),g_0\rangle$ con g_0 el símplice geométrico de dimensión 0, el punto base; o $x = \langle (j_{n-m},\ldots,j_1),g_m\rangle$ con $0 \leq j_1 < \cdots < j_{n-m} \leq n-1$ y g_m el único símplice geométrico de dimensión m. Las propiedades que verifican los conjuntos simpliciales permiten intercambiar el orden de aplicación de un operador cara con otro de degeneración. De esta forma, no se pierde generalidad suponiendo que el primer operador a aplicar sea un operador cara, y, una vez aplicado éste, por la propiedad $\eta^{n+1}_i\eta^n_j = \eta^{n+1}_{j+1}\eta^n_i$ si $i \leq j$, se puede suponer que la lista de degeneraciones es estrictamente decreciente. Aplicando lo anterior, la función se define como sigue: si $(i,n,x) \in Def(\delta_{S^m})$, entonces

a) si i está en la lista de degeneraciones de x, se elimina y se resta 1 a todos los elementos de su izquierda, es decir, a todos los que eran mayores;

b) si i no está en la lista de degeneraciones de x, pero $i-1$ sí está, se elimina de la lista y se resta 1 a todos los elementos de su izquierda;

c) en otro caso, es decir si ni i ni $i-1$ están en la lista de degeneraciones de x, la imagen es el degenerado del punto base en la dimensión correspondiente.

Escribiéndolo formalmente, lo anterior se reduce a:

- $\delta_{S^m}(i, n, \langle (j_{n-m}, \ldots, j_1), g_m \rangle) := \langle (j_{n-m} - 1, \ldots, j_h - 1, j_{h-1}, \ldots, j_1), g_m \rangle$ si $\exists h \in \{1, \ldots, n-m\}$ tal que $j_h = i$ o $j_h = i-1$ con $j_h + 1 \neq i$
- $\delta_{S^m}(i, n, \langle (j_{n-m}, \ldots, j_1), g_m \rangle) := \langle (n-2, \ldots, 1, 0), g_0 \rangle$, en otro caso.

Con la función dimensión que hemos dado para los soportes que siguen el patrón dado por el conjunto I^K_{absmp}, se tiene que S^m es una $\mathtt{CS}$-álgebra.

Para cada conjunto graduado K, denotamos por $\mathcal{C}^K(\mathtt{CS})$ a la subcategoría plena de $\mathcal{C}(\mathtt{CS})$ con objetos las álgebras que tienen a I^K_{absmp} como soporte para el género *smp* y a $\mathbb{N}$ para el género *nat*. En este caso, al seguir el patrón que proporciona I^K_{absmp}, tanto la función dimensión asociada a cada álgebra como el operador de degeneración quedan determinados.

Pasando ya al marco de $\mathtt{CS}_{imp}$-álgebras, denotamos por $\mathcal{C}^K(\mathtt{CS}_{imp})$ a la subcategoría plena de $\mathcal{C}(\mathtt{CS}_{imp})$ con objetos los que tienen a I^K_{absmp} como soporte para el género *smp* y a $\mathbb{N}$ para el género *nat*. El comentario anterior puede extenderse a esta categoría por lo que cada álgebra de $\mathcal{C}^K(\mathtt{CS}_{imp})$ también tiene determinado el operador de degeneración, que no depende del género $imp_{\mathtt{CS}}$ y viene dado por la función η_{I^K}. Por tanto, para definir álgebras de la categoría $\mathcal{C}^K(\mathtt{CS}_{imp})$ es suficiente con determinar el soporte para el género distinguido $imp_{\mathtt{CS}}$ y la interpretación del operador imp_δ.

Otra vez, el hecho de fijar un conjunto graduado como conjunto de símplices geométricos para cada conjunto simplicial resulta excesivamente restrictivo. Hace que no sea posible agrupar en una de nuestras familias de $\mathtt{CS}$-álgebras, en una $\mathtt{CS}_{imp}$-álgebra, conjuntos simpliciales con distintos símplices geométricos. Sin embargo, es posible relajar la condición anterior exigiendo solamente que todos los símplices geométricos de todos los conjuntos simpliciales estén en el conjunto graduado. Considerando esta última situación, tendrá que ser posible extender de modo coherente los operadores cara y degeneración a todos los elementos del conjunto graduado fijo, de forma que dé lugar al mismo conjunto simplicial. Así, volviendo al ejemplo de las esferas, podemos considerar el conjunto graduado $K_\cup = \bigcup_{m\in\mathbb{N}} K_m$ siendo, para cada $n \in \mathbb{N}$, $K^n_\cup = \{g_n\}$, es decir, un único símplice geométrico en cada dimensión. Así, los conjuntos simpliciales que vamos a poder representar tendrán, a lo más, un símplice geométrico en cada dimensión. Éste es el caso de la esfera S^m para cualquier $m \in \mathbb{N}$. Tomamos $\mathbb{N}$ como soporte para *nat* y el soporte de la correspondiente álgebra inicial $I^{K_\cup}_{absmp}$ como soporte para el género *smp*. Observar que, en dimensión n, los símplices degenerados pueden ser de dos tipos:

- Degeneraciones de un símplice geométrico de dimensión h, con $h > 0$: $\langle (j_{n-h}, \ldots, j_1), g_h \rangle$ con $n, h \in \mathbb{N}$, $h < n$ y $0 \leq j_1 < \cdots < j_{n-h} \leq n-1$
- Degeneraciones del punto base (el geométrico de dimensión 0): $\langle (n-1, n-2, \ldots, 1, 0), g_0 \rangle$

La operación $\delta_{S^m}\colon \mathbb{N}\times\mathbb{N}\times I^{K_\cup}_{absmp} \to I^{K_\cup}_{absmp}$ tiene por dominio el conjunto $\big\{(i,n,\langle(j_k,\ldots,j_1),a\rangle) \in \mathbb{N}\times\mathbb{N}\times I^{K_\cup}_{absmp} \mid i\leq n,\ \ n>0$ y $\exists\ l\in\mathbb{N}$ tal que $a\in K^l_\cup$ con $k+l=n\big\}$. En este caso concreto, si $(i,n,x)\in Def(\delta_{S^m})$, entonces hay tres posibilidades para x: $x=\langle(n-1,\ldots,0),g_0\rangle$; $x=\langle(j_{n-m},\ldots,j_1),g_m\rangle$ con $0\leq j_1<\cdots<j_{n-m}\leq n-1$; o $x=\langle(j_{n-l},\ldots,j_1),g_l\rangle$ con $0\leq j_1<\cdots<j_{n-l}\leq n-1$ y $l\notin\{0,m\}$. Para cada $(i,n,x)\in Def(\delta_{S^m})$, la función δ_{S^m} se define del siguiente modo:

- $\delta_{S^m}(i,n,\langle(j_{n-l},\ldots,j_1),g_l\rangle)$:= $\langle(j_{n-l}-1,\ldots,j_h-1,j_{h-1},\ldots,j_1),g_l\rangle$ si $\exists h\in\{1,\ldots,n-l\}$ tal que $j_h=i$ o $j_h=i-1$ siendo $j_h+1\neq i$
- $\delta_{S^m}(i,n,\langle(j_{n-l},\ldots,j_1),g_l\rangle)$:= $\langle(n-2,\ldots,1,0),g_0\rangle$, en otro caso.

Observar que, así definido, el operador cara no depende de la esfera que se esté representando, es el mismo para todas, y vamos a denotarlo por δ_S. El operador de degeneración viene dado por la función $\eta_{I^{K_\cup}}$

Definimos a continuación una $\mathtt{CS}_{imp}$-álgebra de la categoría $\mathcal{C}^{K_\cup}(\mathtt{CS}_{imp})$ que representa a la familia de esferas $\{S^m\}_{m\in\mathbb{N}^*}$. La denotamos por A y viene dada por:

- $A_{imp_{\mathtt{CS}}}=\mathbb{N}^*$
- La función $imp\_\delta_A\colon \mathbb{N}^*\times\mathbb{N}\times\mathbb{N}\times I^{K_\cup}_{absmp}\to I^{K_\cup}_{absmp}$ que proviene del operador cara, tiene por dominio $Def(imp\_\delta_A)=\big\{(m,i,n,\langle(j_k,\ldots,j_1),a\rangle)\in\mathbb{N}^*\times\mathbb{N}\times\mathbb{N}\times I^{K_\cup}_{absmp} \mid i\leq n,\ n>0$ y $\exists\ l\in\mathbb{N}$ tal que $a\in K^l_\cup$ con $k+l=n\big\}$, y se define como $imp\_\delta_A(m,i,n,x):=\delta_S(i,n,x)$.

Es claro que la $\mathtt{CS}_{imp}$-álgebra A representa un ramo de esferas $\{S^m\}_{m\in\mathbb{N}^*}$, todas ellas con el mismo punto base.

Si deseamos representar las esferas anteriores pero colgadas de distinto punto base, podemos considerar el conjunto graduado $K_\sqcup=\{K^n_\sqcup\}_{n\in\mathbb{N}}$ con $K^0_\sqcup=\mathbb{N}$ y $K^n_\sqcup=\{g_n\}$, para todo $n\in\mathbb{N}^*$. Tomamos la $\mathtt{CS}_{imp}$-álgebra B con conjuntos soportes $\mathbb{N}$ e $I^{K_\sqcup}_{absmp}$ para nat y smp, $\mathbb{N}^*$ para $imp_{\mathtt{CS}}$ y con el operador cara definido como sigue: $imp\_\delta_B\colon \mathbb{N}^*\times\mathbb{N}\times\mathbb{N}\times I^{K_\sqcup}_{absmp}\to I^{K_\sqcup}_{absmp}$ con dominio $Def(imp\_\delta_B)=\big\{(m,i,n,\langle(j_k,\ldots,j_1),a\rangle)\in\mathbb{N}^*\times\mathbb{N}\times\mathbb{N}\times I^{K_\sqcup}_{absmp} \mid i\leq n,\ n>0$ y $\exists\ l\in\mathbb{N}$ tal que $a\in K^l_\sqcup$ con $k+l=n\big\}$, y dado por:

- $imp\_\delta_B(m,i,n,\langle(n-1,\ldots,0),k\rangle)$:= $\langle(n-2,\ldots,1,0),m\rangle$
- $imp\_\delta_B(m,i,n,\langle(j_{n-l},\ldots,j_1),g_l\rangle)$:= $\langle(j_{n-l}-1,\ldots,j_h-1,j_{h-1},\ldots,j_1),g_l\rangle$ si $\exists\ h\in\{1,\ldots,n-l\}$ tal que $j_h=i$ o $j_h=i-1$ siendo $j_h+1\neq i$
- $imp\_\delta_B(m,i,n,\langle(j_{n-l},\ldots,j_1),g_l\rangle)$:= $\langle(n-2,\ldots,1,0),m\rangle$, en otro caso.

La $\mathtt{CS}_{imp}$-álgebra B representa a una familia de esferas, una en cada dimensión todas ellas con distinto punto base.

Observar que variando el conjunto de símplices de dimensión 0, desde el unipuntual hasta $\mathbb{N}$, se puede dar toda una gama de ejemplos de familias de esferas, "familias intermedias" entre los dos ejemplos anteriores.

Para cada conjunto graduado K, hemos considerado la categoría $\mathcal{C}^K(\mathtt{CS}_{imp})$ de $\mathtt{CS}_{imp}$-álgebras con soportes I^K_{absmp} y $\mathbb{N}$ para *smp* y *nat* respectivamente. Según los resultados teóricos alcanzados, la subcategoría $\mathcal{C}^{K,\{\}}(\mathtt{CS}_{imp})$, con los mismos objetos y morfismos los que son identidades sobre I^K_{absmp} y $\mathbb{N}$, posee objeto final. Tal y como se ha visto a lo largo del capítulo, una descripción del álgebra final posee como soporte para el género distinguido un espacio de pares de funciones, álgebra que denotamos por $\mathbb{1}^K$. Ahora bien, según hemos comentado, la definición del operador imp_η viene determinada para todas las álgebras de la categoría $\mathcal{C}^K(\mathtt{CS}_{imp})$. Esto nos permite considerar otra descripción del objeto final en la que el soporte asociado al género distinguido es más reducido, está formado por un espacio de funciones. Así, denotamos por $\mathbb{1}^{K,can}$ al álgebra de $\mathcal{C}^K(\mathtt{CS}_{imp})$ definida como sigue:

- $\mathbb{1}^{K,can}_{imp_{\mathtt{CS}}} = \{d\colon \mathbb{N}\times\mathbb{N}\times I^K_{absmp} \to I^K_{absmp} \mid \langle I^K_{absmp}, \mathbb{N}, \{d, \eta_{I^K}\}\rangle \text{ es conjunto simplicial }\}$; es decir, $\langle I^K_{absmp}, \mathbb{N}, \{d, \eta_{I^K}\}\rangle$ es un objeto de la categoría $\mathcal{C}(\mathtt{CS})$.

- $imp\_\delta_{\mathbb{1}^{K,can}}\colon \mathbb{1}^{K,can}_{imp_{\mathtt{CS}}}\times\mathbb{N}\times\mathbb{N}\times I^K_{absmp} \to I^K_{absmp}$ función parcial con dominio el conjunto $\{(d,i,n,x) \in \mathbb{1}^{K,can}_{imp_{\mathtt{CS}}}\times\mathbb{N}\times\mathbb{N}\times I^K_{absmp} \mid (i,n,x)\in Def(d)\}$ y definida, para cada $(d,i,n,x)\in Def(imp\_\delta_{\mathbb{1}^{K,can}})$ por $imp\_\delta_{\mathbb{1}^{K,can}}(d,i,n,x) := d(i,n,x)$.

- $imp\_\eta_{\mathbb{1}^{K,can}}\colon \mathbb{1}^{K,can}_{imp_{\mathtt{CS}}}\times\mathbb{N}\times\mathbb{N}\times I^K_{absmp} \to I^K_{absmp}$ es la función con dominio $\mathbb{1}^{K,can}_{imp_{\mathtt{CS}}}\times Def(\eta_{I^K})$ y definida, para cada $(d,i,n,x)\in Def(imp\_\eta_{\mathbb{1}^{K,can}})$ por $imp\_\eta_{\mathbb{1}^{K,can}}(d,i,n,x) := \eta_{I^K}(i,n,x)$.

Como es de esperar, se tiene el siguiente resultado:

Teorema 2.9.3 *$\mathbb{1}^K$ y $\mathbb{1}^{K,can}$ son álgebras isomorfas en la categoría $\mathcal{C}^{K,\{\}}(\mathtt{CS}_{imp})$.*

El isomorfismo existente entre ambos objetos asegura que es indistinto trabajar con cualquiera de ellos, y que ambos recogen a todos los conjuntos simpliciales con símplices geométricos los elementos de K. Concretamente la descripción de $\mathbb{1}^{K,can}$ se acerca más al modo de trabajo de EAT.

De este ejemplo, podemos deducir que al trabajar con Σ_{imp}-álgebras, una vez fijados los soportes para los géneros de la signatura Σ y hechas explícitas las condiciones exigidas a los modelos, pueden existir operadores imp_σ cuya definición no dependa del género distinguido imp_Σ. En ese caso, estos operadores pueden considerarse visibles e incluirse directamente en la signatura tomada para generar los soportes asociados a los géneros de Σ. Haciéndolo así, pueden verse como operadores externos a las Σ_{imp}-álgebras. En ese caso, no influye en nada el incluirlos o no en la definición del objeto final, es suficiente con disponer de la operación aparte, quedando la descripción funcional del objeto final con menos funciones que el número de operadores de la signatura.

Volvemos a la categoría de $\mathtt{CS}$-álgebras $\mathcal{C}^K(\mathtt{CS})$ y vamos a señalar otra propiedad que verifican todas las álgebras de esa categoría: en cualquier álgebra de la categoría $\mathcal{C}^K(\mathtt{CS})$, conociendo el comportamiento de la función cara sobre los símplices geométricos, se puede determinar el comportamiento de la función sobre los símplices abstractos. Además, la forma de hacerlo es común para todas las álgebras de la categoría. En efecto, para evaluar un operador cara sobre un símplice abstracto (degenerado o no), aplicando las propiedades que permiten intercambiar el orden de aplicación de un operador cara con otro de degeneración, se puede suponer que lo primero que se aplica es un operador cara, y que se aplica sobre un símplice abstracto no

degenerado. Así, basta disponer de una función $f_\delta\colon \mathbb{N}\times\mathbb{N}\times K \to I^K_{absmp}$. Observar que la función f_δ trabaja únicamente sobre símplices geométricos y devuelve símplices abstractos, que pueden ser degenerados o no. El resultado de aplicar esta función es un símplice abstracto al que se le añade la lista de degeneraciones que teníamos (es decir, se le aplican los distintos operadores de degeneración que quedaban pendientes de aplicar). Finalmente, aplicando la propiedad $\eta_i^{n+1}\eta_j^n = \eta_{j+1}^{n+1}\eta_i^n$ si $i \leq j$, se consigue que la lista de degeneraciones sea estrictamente decreciente.

Esta propiedad que verifican las álgebras de la categoría $\mathcal{C}^K(\mathtt{CS})$ respecto al operador δ, se traslada a las álgebras de la categoría $\mathcal{C}^K(\mathtt{CS}_{imp})$ respecto al operador imp_δ. Así, si A es un álgebra de $\mathcal{C}^K(\mathtt{CS}_{imp})$, para obtener la función de imp_δ_A podemos apoyarnos en que cada operador cara de un conjunto simplicial puede determinarse a partir de su definición sobre el conjunto de símplices geométricos, extendiéndose posteriormente a los símplices abstractos tal y como hemos explicado, de forma que se tiene la definición completa de la función.

Por ejemplo, en el caso de la $\mathtt{CS}_{imp}$-álgebra A (definida en la página 111) que recoge a una familia de esferas (todas con el mismo punto base), la función $imp\_f_\delta\colon \mathbb{N}^*\times\mathbb{N}\times\mathbb{N}\times K \to I^K_{absmp}$ con dominio $\{(m,i,n,g) \mid g \in K^n\}$, se define únicamente sobre símplices geométricos del siguiente modo: $imp\_f_\delta(m,i,n,g) := \langle(n-1,\ldots,0),g_0\rangle$. La extensión a los símplices abstractos da lugar a la función imp_δ_A. La definición de la función imp_f_δ proviene de la función $f_\delta\colon \mathbb{N}\times\mathbb{N}\times K \to I^K_{absmp}$ con dominio $\{(i,n,g) \mid g \in K^n\}$ y definida por $f_\delta(i,n,g) := \langle(n-1,\ldots,0),g_0\rangle$, función que define el comportamiento del operador cara i-ésima en dimensión n de la esfera de cualquier dimensión sobre los símplices geométricos.

Los comentarios anteriores permiten dar otra descripción funcional del objeto final de la categoría $\mathcal{C}^{K,\{\}}(\mathtt{CS}_{imp})$, descripción más reducida que las anteriores. Denotamos por $\mathbb{1}^{K,can,gen}$ a la siguiente $\mathtt{CS}_{imp}$-álgebra:

- $\mathbb{1}^{K,can,gen}_{imp_{\mathtt{CS}}} = \left\{d\colon \mathbb{N}\times\mathbb{N}\times K \to I^K_{absmp} \mid \langle I^K_{absmp},\mathbb{N},\{\tilde{d},\eta_{I^K}\}\rangle \text{ es conjunto simplicial}\right\}$, donde $\tilde{d}$ es la extensión *canónica* de la aplicación d al espacio $\mathbb{N}\times\mathbb{N}\times I^K_{absmp}$, extensión que se realiza tal y como se ha explicado aplicando algunas de las propiedades de los conjuntos simpliciales.
- $imp\_\delta_{\mathbb{1}^{K,can,gen}}\colon \mathbb{1}^{K,can,gen}_{imp_{\mathtt{CS}}}\times\mathbb{N}\times\mathbb{N}\times I^K_{absmp} \to I^K_{absmp}$ es la función parcial con dominio $\left\{(d,i,n,x) \in \mathbb{1}^{K,can,gen}_{imp_{\mathtt{CS}}}\times\mathbb{N}\times\mathbb{N}\times I^K_{absmp} \mid (i,n,x) \in Def(\tilde{d})\right\}$ definida, para cada $(d,i,n,x) \in Def(imp\_\delta_{\mathbb{1}^{K,can,gen}})$ por $imp\_\delta_{\mathbb{1}^{K,can,gen}}(d,i,n,x) := \tilde{d}(i,n,x)$.
- La operación $imp\_\eta_{\mathbb{1}^{K,can,gen}}$ se define como la del objeto $\mathbb{1}^{K,can}$.

El siguiente teorema recoge la propiedad de finalidad de este objeto en la categoría $\mathcal{C}^{K,\{\}}(\mathtt{CS}_{imp})$.

Teorema 2.9.4 *$\mathbb{1}^K$, $\mathbb{1}^{K,can}$ y $\mathbb{1}^{K,can,gen}$ son álgebras isomorfas en la categoría $\mathcal{C}^{K,\{\}}(\mathtt{CS}_{imp})$.*

De las tres descripciones vistas para el objeto final en el caso de los conjuntos simpliciales, ésta última es la que más cerca está del modo de trabajo de EAT.

Este ejemplo pretende poner de manifiesto dos cuestiones. Por una parte, que cuando los elementos de un conjunto soporte de un álgebra siguen un patrón, hay casos en los que es posible que algunas operaciones puedan considerarse visibles. Por otra parte, cuando un

conjunto soporte de un álgebra posee alguna propiedad de generabilidad (entendida en sentido amplio), no es necesario disponer explícitamente en el objeto final del comportamiento de las operaciones sobre todos los elementos del soporte, basta con conocerlo sobre los generadores. En cualquiera de estos casos, los correspondientes objetos finales de las categorías de familias de álgebras admiten descripciones más sencillas.

Por tanto, hemos abordado dos posibles reducciones en la descripción de los objetos finales como tuplas de funciones: en el número de funciones y en los dominios de éstas.

2.9.3 Complejos de cadenas

Sea R un anillo. Un *complejo de cadenas* $C = \{C_p, d_p\}_{p\in\mathbb{Z}}$ es una familia de R-módulos $\{C_p\}_{p\in\mathbb{Z}}$ junto con una familia de homomorfismos de R-módulos $\{d_p\}_{p\in\mathbb{Z}}$, $d_p\colon C_p \to C_{p-1}$, verificando que $d_{p-1} \circ d_p = 0$, para cada $p \in \mathbb{Z}$. Los operadores $\{d_p\}_{p\in\mathbb{Z}}$ son conocidos como diferenciales y la propiedad que satisfacen se suele escribir $d^2 = 0$. Si $C = \{C_p, d_p\}_{p\in\mathbb{Z}}$ y $C' = \{C'_p, d'_p\}_{p\in\mathbb{Z}}$ son complejos de cadenas, un *morfismo* $f\colon C \to C'$ entre ellos, es una familia de homomorfismos de módulos, $\{f_p\colon C_p \to C'_p\}_{p\in\mathbb{Z}}$, tales que $f_{p-1} \circ d_p = d'_p \circ f_p$, para todo $p \in \mathbb{Z}$.

Los complejos de cadenas junto con los morfismos entre ellos forman una categoría que denominamos Categoría de Complejos de Cadenas. Las definiciones anteriores están extraídas de [73] (página 39) y son definiciones habituales en Álgebra Homológica y Topología Algebraica.

Conviene hacer notar que EAT no trabaja con todos los tipos de complejos de cadenas, solamente lo hace con complejos de cadenas formados por $\mathbb{Z}$-módulos libres (véase en [104] el capítulo 1). Éste es el motivo por el que consideraremos únicamente estos últimos. Por tanto, en lo que sigue, un *complejo de cadenas* $C = \{C_p, d_p\}_{p\in\mathbb{Z}}$ será una familia de $\mathbb{Z}$-módulos libres $\{C_p\}_{p\in\mathbb{Z}}$ junto con una familia $\{d_p\}_{p\in\mathbb{Z}}$ de morfismos de $\mathbb{Z}$-módulos, $d_p\colon C_p \to C_{p-1}$, (*aplicaciones diferenciales*), tales que $d_{p-1} \circ d_p = 0$, para cada $p \in \mathbb{Z}$.

Un $\mathbb{Z}$-módulo C_p no es sino un grupo abeliano donde se señala como operación externa el producto $mult\colon \mathbb{Z} \times C_p \to C_p$ con el significado habitual. Es decir, $mult(m, a)$ es $suma(a, suma(a, \dots))\dots)$ (m veces) siendo *suma* la operación binaria de grupo (ver [73], página 10). Por otro lado, el que C_p sea $\mathbb{Z}$-módulo libre implica que, en particular, es generado, por lo que cada objeto de C_p puede expresarse como una combinación lineal sobre el conjunto de generadores. Así, a los elementos de C_p se les denomina *combinaciones de grado* p. Cada grupo C_p es entonces una suma directa $\mathbb{Z} \oplus \cdots \oplus \mathbb{Z} \oplus \dots$ de tantas copias de $\mathbb{Z}$ como generadores tenga el grupo abeliano C_p.

Con el objetivo de obtener una representación para los complejos de cadenas usados en EAT vamos a definir un TAD, esto es, una signatura y una categoría de álgebras, que represente a los complejos de cadenas cuyas componentes son $\mathbb{Z}$-módulos libres.

Teniendo en cuenta que la base de un complejo de cadenas es un grupo graduado, consideramos un único género *cmbn* en el que se recogerán los elementos de todos los grupos, junto con otro género, el género *int*, asociado a la graduación. Si se quisieran considerar complejos de cadenas sobre otros anillos distintos a $\mathbb{Z}$, habría que tener en cuenta al anillo y a la operación externa del módulo (en particular, habría que considerar otro género para los elementos del anillo). Como operaciones, la signatura debe recoger las correspondientes a $\mathbb{Z}$-módulo y las diferenciales. Así, definimos la signatura `CC` como la signatura con géneros *int* y *cmbn* y con las siguientes operaciones:

$$\begin{array}{lcccc} suma & : & cmbn\ \ cmbn & \to & cmbn \\ opp & : & cmbn & \to & cmbn \\ zero & : & int & \to & cmbn \\ mult & : & int\ \ cmbn & \to & cmbn \\ dffr & : & cmbn & \to & cmbn \end{array}$$

No nos interesa considerar todas las CC-álgebras sino solo aquellas que representan complejos de cadenas formados por $\mathbb{Z}$-módulos libres. Por ello, debemos considerar una subcategoría adecuada de $PAlg(\mathtt{CC})$. Denotamos por $\mathcal{C}(\mathtt{CC})$ a la subcategoría de $PAlg(\mathtt{CC})$ cuyos objetos son las CC-álgebras A que verifican las siguientes condiciones:

i) El conjunto soporte para el género *int* es $\mathbb{Z}$.

ii) Existe una función total $grado_A\colon A_{cmbn} \to \mathbb{Z}$ que asocia a cada elemento de A_{cmbn} un grado.

iii) Las operaciones $opp_A, zero_A, mult_A$ y $dffr_A$ son totales.

iv) $Def(suma_A) = \{(x,y) \in A_{cmbn} \times A_{cmbn} \mid grado_A(x) = grado_A(y)\}$

v) Para cada $p \in \mathbb{Z}$, si consideramos el conjunto $A_p = \{x \in A_{cmbn} \mid grado_A(x) = p\}$, el álgebra $\langle A_p, \{suma_A|_{A_p}, opp_A|_{A_p}, zero_A|_{\{p\}}\rangle$ es grupo abeliano.

vi) Para cada $p \in \mathbb{Z}$, el módulo en ese grado debe ser libre. Esto significa que debe existir G_p subconjunto de A_p de forma que cualquier elemento de A_p pueda descomponerse, y de forma única, como una combinación lineal con coeficientes en $\mathbb{Z}$ de elementos de G_p. Es decir, para cada $p \in \mathbb{Z}$ y cada $x \in A_p$, existe un conjunto $\{(m_1,g_1),\ldots,(m_n,g_n)\}$ con $(m_1,\ldots,m_n) \in \mathbb{Z}^n$, $(g_1,\ldots,g_n) \in G_p^n$, únicos si $m_i \neq 0$ para todo $i = 1,\ldots,n$ y $g_i \neq g_j$, para todo $i \neq j$, tales que $x = suma_A(mult_A(m_1,g_1), suma_A(mult_A(m_2,g_2),\ldots,suma_A(mult_A(m_n,g_n), zero_A(p))\ldots))$.

vii) La función $grado_A$ debe ser compatible con las operaciones, es decir, debe verificar:

- para cada par $(x,y) \in Def(suma_A)$, $grado_A(suma_A(x,y)) = grado_A(x) = grado_A(y)$;
- para cada $x \in A_{cmbn}$, $grado_A(opp_A(x)) = grado_A(x)$;
- para cada $p \in \mathbb{Z}$, $grado_A(zero_A(p)) = p$;
- para cada $i \in \mathbb{Z}$ y cada $x \in A_{cmbn}$, $grado_A(mult_A(i,x)) = grado_A(x)$;
- para cada $x \in A_{cmbn}$, $grado_A(dffr_A(x)) = grado_A(x) - 1$.

viii) La función $dffr_A$ debe ser compatible con el resto de operaciones (con la idea de que la restricción a cada A_p sea homomorfismo de $\mathbb{Z}$-módulos), por lo que debe verificarse:

- $dffr_A(suma_A(x,y)) = suma_A(dffr_A(x), dffr_A(y))$, para cada par $(x,y) \in Def(suma_A)$;

- para cada $x \in A_{cmbn}$, $dffr_A(opp_A(x)) = opp_A(dffr_A(x))$;
- para cada $p \in \mathbb{Z}$, $dffr_A(zero_A(p)) = zero_A(p-1)$;
- para cada $i \in \mathbb{Z}$ y cada $x \in A_{cmbn}$, $dffr_A(mult_A(i,x)) = mult_A(i, dffr_A(x))$.

ix) Para cada $x \in A_{cmbn}$ con $grado_A(x) = p$, se tiene que $dffr_A(dffr_A(x)) = zero_A(p-2)$

Como morfismos de la categoría $\mathcal{C}(\mathtt{CC})$ tomamos aquellos que son la identidad sobre $\mathbb{Z}$ y que respetan el grado de los elementos.

Notar que las propiedades que debe cumplir la operación externa de un módulo no han sido explícitamente impuestas. Esto es así porque estamos con módulos sobre $\mathbb{Z}$, es decir, basta exigir que sean grupos abelianos.

Es fácil ver que la categoría $\mathcal{C}(\mathtt{CC})$ es equivalente a la subcategoría de la categoría de complejos de cadenas con objetos los complejos de cadenas formados por $\mathbb{Z}$-módulos libres. La clave para la existencia de la equivalencia entre ambas categorías es de nuevo, al igual que en los ejemplos anteriores, la existencia de la función *grado*. Así, ambas categorías tienen el mismo poder expresivo y utilizaremos $\mathcal{C}(\mathtt{CC})$ para continuar nuestra formalización del modo en el que los complejos de cadenas son tratados en EAT.

Consideramos el TAD $\mathcal{T}_{\mathtt{CC}} = \langle \mathtt{CC}, \mathcal{C}(\mathtt{CC}) \rangle$, que denominamos *TAD de los Complejos de Cadenas*. Para continuar, aplicamos la operación $()_{imp}$ a $\mathcal{T}_{\mathtt{CC}}$ y obtenemos el correspondiente TAD $\mathcal{T}_{\mathtt{CC}_{imp}} = \langle \mathtt{CC}_{imp}, \mathcal{C}(\mathtt{CC}_{imp}) \rangle$ que recoge a las familias de complejos de cadenas de la categoría $\mathcal{C}(\mathtt{CC})$. La signatura $\mathtt{CC}_{imp}$ es la signatura con géneros *int*, *cmbn* e $imp_{\mathtt{CC}}$ y con las siguientes operaciones

$$
\begin{array}{rcllll}
imp\_suma & : & imp_{\mathtt{CC}} & cmbn & cmbn & \rightarrow & cmbn \\
imp\_opp & : & imp_{\mathtt{CC}} & cmbn & & \rightarrow & cmbn \\
imp\_zero & : & imp_{\mathtt{CC}} & int & & \rightarrow & cmbn \\
imp\_mult & : & imp_{\mathtt{CC}} & int & cmbn & \rightarrow & cmbn \\
imp\_dffr & : & imp_{\mathtt{CC}} & cmbn & & \rightarrow & cmbn
\end{array}
$$

Por su parte, $\mathcal{C}(\mathtt{CC}_{imp})$ es la subcategoría plena de $PAlg(\mathtt{CC}_{imp})$ con objetos las $\mathtt{CC}_{imp}$-álgebras $A = \langle A_{imp_{\mathtt{CC}}}, A_{cmbn}, \mathbb{Z}, \{(imp\_suma_A, Def(imp\_suma_A)), (imp\_opp_A, Def(imp\_opp_A)), (imp\_zero_A, Def(imp\_zero_A)), (imp\_mult_A, Def(imp\_mult_A)), (imp\_dffr_A, Def(imp\_dffr_A))\} \rangle$, tales que, para cada $a \in A_{imp_{\mathtt{CC}}}$, la $\mathtt{CC}$-álgebra $A_a = \langle A_{cmbn}, \mathbb{Z}, \{(imp\_suma_A(a,-), Def(imp\_suma_A(a,-))), (imp\_opp_A(a,-), Def(imp\_opp_A(a,-))), (imp\_zero_A(a,-), Def(imp\_zero_A(a,-))), (imp\_mult_A(a,-), Def(imp\_mult_A(a,-))), (imp\_dffr_A(a,-), Def(imp\_dffr_A(a,-)))\} \rangle$ está en $\mathcal{C}(\mathtt{CC})$.

Otra vez, cada objeto de la categoría $\mathcal{C}(\mathtt{CC}_{imp})$ representa a una familia indexada, en este caso, de complejos de cadenas.

Este ejemplo va a presentar similitudes claras con el de los conjuntos simpliciales, sobre todo en lo referente a graduación y generabilidad. A partir de aquí fijamos los conjuntos soporte para los géneros de la signatura $\mathtt{CC}$. Para el género *int* ya está fijo y es $\mathbb{Z}$. Para el género *cmbn* vamos

a definirlo, siguiendo lo estudiado en el caso de los conjuntos simpliciales, como el soporte del álgebra inicial para una especificación concreta. En este caso, ya que cualquier elemento puede expresarse como combinación lineal de algunos de los generadores, partiremos de un conjunto graduado $G = \{G_p\}_{p\in\mathbb{Z}}$ que recoja a los generadores. Sea $G = \{G_p\}_{p\in\mathbb{Z}}$ un conjunto graduado fijo y nuestro objetivo es representar familias de complejos de cadenas cuyos generadores son los elementos de G. Para ello, comenzamos considerando una signatura cuya álgebra inicial (uno de sus conjuntos soporte) utilizaremos como soporte para el género *cmbn*. Consideramos la signatura `CMBN` con géneros *int*, *gnr* y *cmbn* y con las siguientes operaciones

$$\begin{array}{lcccccc}
coer & : & gnr & & & \to & cmbn \\
suma & : & cmbn & cmbn & & \to & cmbn \\
add & : & int & gnr & cmbn & \to & cmbn \\
opp & : & cmbn & & & \to & cmbn \\
zero & : & int & & & \to & cmbn \\
mult & : & int & cmbn & & \to & cmbn \\
grado & : & cmbn & & & \to & int
\end{array}$$

Se pretende que el género *gnr* represente a los generadores y *cmbn* a la representación de cualquier elemento como combinación lineal de generadores con coeficientes en $\mathbb{Z}$. La idea de la operación *coer* es construir la representación como combinación de cada generador. Por su parte, la operación *add* servirá para sumar un monomio a una combinación lineal, lo que determinará la suma de combinaciones. Observar que, a diferencia del ejemplo de los conjuntos simpliciales, hemos incluido en la signatura explícitamente la operación *grado*, en lugar de exigir su existencia a las álgebras. Lo hemos hecho así, para dejar más patente que es indistinto hacerlo de un modo u otro. También en este ejemplo podemos suponer que la representación de cada elemento de cada $\mathbb{Z}$-módulo de un complejo de cadenas está determinada, es un par formado por el índice del $\mathbb{Z}$-módulo al que pertenece y una combinación lineal con coeficientes en $\mathbb{Z}$ de generadores de ese $\mathbb{Z}$-módulo. El hecho de disponer de la representación de cada elemento nos permite conocer el comportamiento de los operadores de cada $\mathbb{Z}$-módulo, por lo que los hemos incluido en la signatura anterior. Además, a la hora de considerar una subcategoría adecuada de álgebras para la signatura anterior, podremos imponer las condiciones de $\mathbb{Z}$-módulo.

Lo anterior nos lleva a considerar la categoría con objetos las `CMBN`-álgebras A que verifican las siguientes condiciones:

i) El conjunto soporte para el género *int* es $\mathbb{Z}$.

ii) El conjunto soporte para el género *gnr* es $\bigsqcup_{p\in\mathbb{Z}} G_p$.

iii) Las funciones $grado_A, coer_A, opp_A, zero_A$ y $mult_A$ son totales.

iv) $Def(suma_A) = \{(x, y) \in A_{cmbn} \times A_{cmbn} \mid grado_A(x) = grado_A(y)\}$

v) $Def(add_A) = \{(i, g, x) \in \mathbb{Z} \times A_{gnr} \times A_{cmbn} \mid grado_A(x) = grado_A(coer_A(g))\}$

vi) La función $grado_A$ y las operaciones son compatibles, es decir, se verifica:

- $grado_A(coer_A(x)) = p$ siendo p tal que $x \in G_p$;
- para cada par $(x, y) \in Def(suma_A)$, se verifica que $grado_A(suma_A(x, y)) = grado_A(x) = grado_A(y)$;
- para cada terna $(i, g, x) \in Def(add_A)$, se tiene que $grado_A(add_A(i, g, x)) = grado_A(x) = grado_A(coer_A(g))$;
- para cada $x \in A_{cmbn}$, $grado_A(opp_A(x)) = grado_A(x)$;
- para cada $i \in \mathbb{Z}$, $grado_A(zero_A(i)) = i$;
- para cada $i \in \mathbb{Z}$ y cada $x \in A_{cmbn}$, $grado_A(mult_A(i, x)) = grado_A(x)$.

vii) Las funciones $suma_A$ y add_A son compatibles, es decir: $add_A(i, g, x) = suma_A(mult_A(i, coer_A(g)), x)$.

viii) La operación add_A es conmutativa, es decir, $add_A(i_1, g_1, add_A(i_2, g_2, x)) = add_A(i_2, g_2, add_A(i_1, g_1, x))$, para todo $i_1, i_2 \in \mathbb{Z}$, para todo $g_1, g_2 \in G_p$ para algún $p \in \mathbb{Z}$ y para todo $x \in A_{cmbn}$ tal que $grado_A(x) = p$.

ix) Para cada $p \in \mathbb{Z}$, consideramos el conjunto $A_p = \{x \in A_{cmbn} \mid grado_A(x) = p\}$ y se tiene que $\langle A_p, \{suma_A|_{A_p}, opp_A|_{A_p}, zero_A|_{\{p\}}\rangle$ es grupo abeliano.

x) Para cada $p \in \mathbb{Z}$ y cada $x \in A_{cmbn}$ con $grado_A(x) = p$, existen $\{(m_1, g_1), \ldots, (m_n, g_n)\}$ con $(m_1, \ldots, m_n) \in \mathbb{Z}^n$, $(g_1, \ldots, g_n) \in G_p^n$, que son únicos si $m_i \neq 0$ para todo $i = 1, \ldots, n$ y $g_i \neq g_j$, para todo $i \neq j$, tales que $x = suma_A(mult_A(m_1, coer_A(g_1)), suma_A(mult_A(m_2, coer_A(g_2)), suma_A(\ldots suma_A(mult_A(m_n, coer_A(g_n)), zero_A(p))\ldots))$.

Al exigir las condiciones anteriores y fijar los soportes para *int* y *gnr* queda determinado el comportamiento de los operadores, de forma que cada álgebra de la categoría es un $\mathbb{Z}$-módulo graduado libre con conjunto de generadores en cada grado el conjunto G_p correspondiente. El operador *add* se ha incluido en la signatura para facilitar la definición del operador *suma*, de ahí las condiciones exigidas entre ambos (podíamos no haberlo incluido y haber añadido las condiciones sobre *suma*, obteniendo una categoría equivalente a la nuestra).

Como morfismos de la categoría tomamos aquellos que son la identidad sobre los soportes correspondientes a los géneros *int* y *gnr* y que respetan los grados.

Un objeto que pertenece a la categoría anterior es el objeto I^G definido como sigue:

- El conjunto soporte para el género *cmbn* es el siguiente

$$I^G_{cmbn} = \Big\{\langle p, [(t_1, g_1), \ldots, (t_m, g_m)]\rangle \mid p \in \mathbb{Z},\ m \in \mathbb{Z},\ m \geq 0,\ t_i \in \mathbb{Z} \text{ con } t_i \neq 0 \text{ para todo } i = 1, \ldots, m,\ g_i \in G_p \text{ para todo } i = 1, \ldots, m \text{ y } g_i \neq g_j \text{ si } i \neq j\Big\}$$

- $coer_{I^G} \colon I^G_{gnr} \to I^G_{cmbn}$ es la función total dada por $coer_{I^G}(g) := \langle p, [(1, g)]\rangle$ siendo $p \in \mathbb{Z}$ tal que $g \in G_p$.
- La función total $grado_{I^G} \colon I^G_{cmbn} \to I^G_{cmbn}$ viene dada como sigue

$$grado_{I^G}\Big(\langle p, [(t_1, g_1), \ldots, (t_m, g_m)]\rangle\Big) := p$$

- $zero_{I^G} \colon \mathbb{Z} \to I^G_{cmbn}$ es la función total definida como $zero_{I^G}(p) := \langle p, [(\,)] \rangle$.
- $opp_{I^G} \colon I^G_{cmbn} \to I^G_{cmbn}$ es la función total dada por

$$opp_{I^G}\Big(\langle p, [(t_1, g_1), \ldots, (t_m, g_m)] \rangle\Big) := \langle p, [(-t_1, g_1), \ldots, (-t_m, g_m)] \rangle$$

- $mult_{I^G} \colon \mathbb{Z} \times I^G_{cmbn} \to I^G_{cmbn}$ es la función total dada por
  - $mult_{I^G}(0, x) := \langle p, [(\,)] \rangle$ con $grado_{I^G}(x) = p$, y
  - $mult_{I^G}(i, \langle p, [(t_1, g_1), \ldots, (t_m, g_m)] \rangle) := \langle p, [(i * t_1, g_1), \ldots, (i * t_m, g_m)] \rangle$, si $i \neq 0$.
- $add_{I^G} \colon \mathbb{Z} \times I^G_{gnr} \times I^G_{cmbn} \to I^G_{cmbn}$ es la función parcial con dominio

$$Def(add_{I^G}) = \big\{(i, g, x) \;\mid\; grado_{I^G}(x) = grado_{I^G}(coer_{I^G}(g))\big\}$$

  y definida como sigue
  - $add_{I^G}(0, g, x) := x$
  - si $i \neq 0$ y $g \neq g_j$ para todo $j = 1, \ldots, m$,

$$add_{I^G}\Big(i, g, \langle p, [(t_1, g_1), \ldots, (t_m, g_m)] \rangle\Big) := \langle p, [(t_1, g_1), \ldots, (t_m, g_m), (i, g)] \rangle$$

  - si $i \neq 0$ y $\exists j \in \{1, \ldots, m\}$ tal que $g = g_j$ siendo $i + t_j \neq 0$, entonces

$$add_{I^G}\Big(i, g, \langle p, [(t_1, g_1), \ldots, (t_m, g_m)] \rangle\Big) := \langle p, [(t_1, g_1), \ldots, (i + t_j, g_j), \ldots, (t_m, g_m)] \rangle$$

  - si $i \neq 0$ y $\exists j \in \{1, \ldots, m\}$ tal que $g = g_j$ con $i + t_j = 0$, entonces
    $add_{I^G}\Big(i, g, \langle p, [(t_1, g_1), \ldots, (t_m, g_m)] \rangle\Big) :=$

$$\langle p, [(t_1, g_1), \ldots, (t_{j-1}, g_{j-1}), (t_{j+1}, g_{j+1}), \ldots, (t_m, g_m)] \rangle$$

- $suma_{I^G} \colon I^G_{cmbn} \times I^G_{cmbn} \to I^G_{cmbn}$ es la función parcial con dominio el conjunto $\big\{(x, y) \in I^G_{cmbn} \times I^G_{cmbn} \;\mid\; grado_{I^G}(x) = grado_{I^G}(y)\big\}$ y definida como sigue:

$$suma_{I^G}\Big(\langle p, [(t_1, g_1), \ldots, (t_m, g_m)] \rangle, y\Big) := add_{I^G}(t_1, g_1, add_{I^G}(t_2, g_2, add_{I^G}(\ldots, add_{I^G}(t_m, g_m, y)) \ldots))$$

Este objeto I^G verifica todas las condiciones impuestas a las álgebras de la categoría definida anteriormente, siendo $G_p^{I^G} = \big\{coer_{I^G}(g) \;\mid\; g \in G_p\big\}$ el conjunto de generadores, para cada $p \in \mathbb{Z}$.

Esta categoría de `CMBN`-álgebras posee objeto inicial que puede describirse como un cociente de I^G. Basta considerar como conjunto soporte para el género *cmbn* el cociente de I^G_{cmbn} por la relación que identifica aquellos elementos del mismo grado y cuyas listas formadas por los pares coeficiente-generador poseen los mismos elementos (aunque puede que en otro orden). Denotamos por $I^G_{\sim}$ a este cociente de I^G_{cmbn} y lo fijamos como soporte para el género *cmbn*. Así, $I^G_{\sim}$ proporciona un patrón para representar combinaciones.

Para cada conjunto graduado $G = \{G_p\}_{p \in \mathbb{Z}}$, nos vamos a restringir a $\mathcal{C}^{G,\{\}}(\mathtt{CC}_{imp})$, subcategoría de $\mathcal{C}(\mathtt{CC}_{imp})$ cuyos objetos tienen como conjuntos soporte $\mathbb{Z}$ e $I^G_{\sim}$, para *int* y *cmbn* respectivamente, y con morfismos los que son identidades sobre $\mathbb{Z}$ e $I^G_{\sim}$. Según lo visto en la

parte teórica del capítulo, la categoría $\mathcal{C}^{G,\{\}}(\mathtt{CC}_{imp})$ posee objeto final, una de cuyas descripciones tiene como soporte para el género $imp_{\mathtt{CC}}$ un espacio funcional, en el que cada elemento es una tupla formada por cinco funciones (una por cada operación de la signatura CC y con la aridad correspondiente). Siguiendo la misma notación que en el resto del capítulo, denotaremos a este objeto por $\mathbb{1}^G$.

Ahora bien, cualquier álgebra A de la categoría $\mathcal{C}^{G,\{\}}(\mathtt{CC}_{imp})$ tiene determinadas las operaciones $imp\_suma_A, imp\_opp_A, imp\_zero_A$ e imp_mult_A. Esto es debido a que, por definición, para cada $a \in A_{imp_{\mathtt{CC}}}$, la CC-álgebra A_a debe ser objeto de $\mathcal{C}^G(\mathtt{CC})$. Esta última condición lleva consigo que, para que todo sea compatible, la función $grado_{A_a}$ debe coincidir con la función $grado_{IG}$, y que si denotamos por $A_p = \{x \in I^G_\sim \mid grado_{A_a}(x) = p\}$, se tiene que $\langle A_p, \{imp\_suma_{A_a}|_{A_p}, imp\_opp_{A_a}|_{A_p}, imp\_zero_{A_a}|_{\{p\}}\}\rangle$ es grupo abeliano libre sobre el conjunto de generadores $G_p^{A_a} = \{\langle p, [(1,g)]\rangle \in I^G_\sim \mid g \in G_p\}$. Además, por definición del objeto I^G, se tiene que $\langle A_p, \{suma_{IG}|_{A_p}, opp_{IG}|_{A_p}, zero_{IG}|_{\{p\}}\}\rangle$ también lo es y sobre el mismo conjunto de generadores. Tenemos dos grupos abelianos libres con los mismos generadores, luego son el mismo. De esta forma, las funciones $imp\_suma_{A_a}, imp\_opp_{A_a}$ e $imp\_zero_{A_a}$ coinciden con las funciones $suma_{IG}, opp_{IG}$ y $zero_{IG}$, respectivamente. Además, por linealidad, $imp\_mult_{A_a}$ coincide con $mult_{IG}$. Por tanto, las funciones $imp\_suma_{A_a}, imp\_opp_{A_a}, imp\_zero_{A_a}$ e $imp\_mult_{A_a}$ no dependen de a, y se tiene que las funciones $imp\_suma_A, imp\_opp_A, imp\_zero_A, imp\_mult_A$ vienen definidas como $suma_{IG}, opp_{IG}, zero_{IG}$ y $mult_{IG}$ respectivamente.

Así, las álgebras de $\mathcal{C}^{G,\{\}}(\mathtt{CC}_{imp})$ tienen determinadas las operaciones correspondientes a $imp\_suma, imp\_opp, imp\_zero$ e imp_mult las que, además, no dependen del argumento $imp_{\mathtt{CC}}$. Por ello, las operaciones anteriores pueden considerarse visibles, lo que hace que la única información necesaria para recuperar un álgebra de $\mathcal{C}^G(\mathtt{CC})$ sea la diferencial. Esto nos permite obtener otra descripción más sencilla del objeto final que denotamos por $\mathbb{1}^{G,can}$ y que viene dada como sigue:

- $\mathbb{1}^{G,can}_{imp_{\mathtt{CC}}} = \{df\colon I^G_\sim \to I^G_\sim \mid \langle I^G_\sim, \mathbb{Z}, \{suma_{IG}, opp_{IG}, zero_{IG}, mult_{IG}, df\}\rangle$ es complejo de cadenas de $\mathcal{C}^G(\mathtt{CC})\}$.
- $imp\_suma_{\mathbb{1}^{G,can}}\colon \mathbb{1}^{G,can}_{imp_{\mathtt{CC}}} \times I^G_\sim \times I^G_\sim \to I^G_\sim$ es la función parcial con dominio $\mathbb{1}^{G,can}_{imp_{\mathtt{CC}}} \times \{(x,y) \mid grado_{IG}(x) = grado_{IG}(y)\}$ definida, para cada $(df, x, y) \in Def(imp\_suma_{\mathbb{1}^{G,can}})$ como $imp\_suma_{\mathbb{1}^{G,can}}(df, x, y) := suma_{IG}(x, y)$.
- $imp\_opp_{\mathbb{1}^{G,can}}\colon \mathbb{1}^{G,can}_{imp_{\mathtt{CC}}} \times I^G_\sim \to I^G_\sim$ es la función total definida como $imp\_opp_{\mathbb{1}^{G,can}}(df, x) := opp_{IG}(x)$.
- $imp\_zero_{\mathbb{1}^{G,can}}\colon \mathbb{1}^{G,can}_{imp_{\mathtt{CC}}} \times \mathbb{Z} \to I^G_\sim$ es la función total dada por $imp\_zero_{\mathbb{1}^{G,can}}(df, p) := zero_{IG}(p)$.
- $imp\_mult_{\mathbb{1}^{G,can}}\colon \mathbb{1}^{G,can}_{imp_{\mathtt{CC}}} \times \mathbb{Z} \times I^G_\sim \to I^G_\sim$ es la función total definida por $imp\_mult_{\mathbb{1}^{G,can}}(df, i, x) := mult_{IG}(i, x)$.
- $imp\_dffr_{\mathbb{1}^{G,can}}\colon \mathbb{1}^{G,can}_{imp_{\mathtt{CC}}} \times I^G_\sim \to I^G_\sim$ es la función total definida por $imp\_dffr_{\mathbb{1}^{G,can}}(df, x) := df(x)$.

Teorema 2.9.5 *$\mathbb{1}^G$ y $\mathbb{1}^{G,can}$ son álgebras isomorfas en la categoría $\mathcal{C}^{G,\{\}}(\mathtt{CC}_{imp})$.*

En este caso, al igual que en el ejemplo de los conjuntos simpliciales, aún podemos simplificar más la descripción funcional del objeto final. Esta terera descripción proviene de utilizar la

propiedad de que los $\mathbb{Z}$-módulos que forman el complejo de cadenas son, en particular, generados y, por tanto, los operadores pueden definirse sobre el conjunto de generadores y extenderse por linealidad al resto. Así, el operador *imp_dffr*, puede definirse a partir de otro operador que trabaje sobre menos elementos (los generadores) y que después se extiende de modo canónico al resto (esta propiedad tiene que ver con la generabilidad al igual que la que verifica el operador *imp_δ* en el caso de los conjuntos simpliciales, aunque en distinto sentido: en el caso de los conjuntos simpliciales proviene de la representación utilizada para los símplices, mientras que en este caso tiene más que ver con las propiedades de los complejos de cadenas como estructuras matemáticas). Así, para cada álgebra A de $\mathcal{C}^G(\mathtt{CC})$, consideramos el conjunto $Gen = \bigcup_{p\in\mathbb{Z}} G_p$ que recoge a los generadores del complejo de cadenas. Cada función parcial $df_A\colon \mathbb{Z}\times Gen \to I^G_\sim$ con dominio $\{(p,g) \mid g \in G_p\}$, puede extenderse, por linealidad, a una función total $\widetilde{df}_A\colon I^G_\sim \to I^G_\sim$ definida como sigue:

$$\widetilde{df}_A\big(\langle p, [(t_1,g_1),\ldots,(t_m,g_m)]\rangle\big) := suma_A(mult_A(t_1, df_A(p,g_1)),\, suma_A(mult_A(t_2, df_A(p,g_2)),\\ \ldots,\, suma_A(mult_A(t_m, df_A(p,g_m)), zero_A(p))\ \ldots\ ))$$

Para que todo sea correcto, habrá que imponer que la función extendida a partir de la básica verifique las condiciones necesarias. En este caso, y abusando de la notación por no complicarla, la condición de la diferencial se traduce en que para cada $p \in \mathbb{Z}$ y para cada $g \in G_p$, debe cumplirse $\widetilde{df}_A(p-1, df(p,g)) = \langle p-2, [(\ )]\rangle$.

Otra cuestión, y de la que no nos ocupamos, es estudiar qué condiciones debería verificar la función df_A para que al extenderla, por linealidad en este caso, se verifiquen las propiedades requeridas.

Por la relación existente entre las categorías $\mathcal{C}^G(\mathtt{CC}_{imp})$ y $\mathcal{C}^G(\mathtt{CC})$, cualquier álgebra de $\mathcal{C}^G(\mathtt{CC}_{imp})$ verifica la propiedad descrita anteriormente y que posee cualquier álgebra de $\mathcal{C}^G(\mathtt{CC})$. Así, para cada A álgebra de $\mathcal{C}^G(\mathtt{CC}_{imp})$, la función imp_dffr_A puede definirse por extensión a partir de una función parcial $imp\_df_A\colon A_{imp_{\mathtt{CC}}}\times\mathbb{Z}\times Gen \to I^G_\sim$ con dominio $\big\{(a,p,g) \mid g \in G_p\big\}$.

Lo anterior significa que para recuperar un álgebra de $\mathcal{C}^G(\mathtt{CC})$ es suficiente con disponer del comportamiento de la diferencial sobre los generadores, lo que nos lleva a definir el siguiente objeto de $\mathcal{C}^G(\mathtt{CC}_{imp})$ que denotamos por $\mathbb{1}^{G,can,gen}$:

- $\mathbb{1}^{G,can,gen}_{imp_{\mathtt{CC}}} = \big\{df\colon \mathbb{Z}\times G \to I^G_\sim \mid \langle I^G_\sim, \mathbb{Z}, \{suma_{IG}, opp_{IG}, zero_{IG}, mult_{IG}, \widetilde{df}\}\rangle$ es complejo de cadenas de $\mathcal{C}^G(\mathtt{CC})\big\}$, siendo $\widetilde{df}$ la extensión definida por linealidad de la aplicación df a $I^G_\sim$.
- Las funciones $imp\_suma_{\mathbb{1}^{G,can,gen}}$, $imp\_opp_{\mathbb{1}^{G,can,gen}}$, $imp\_zero_{\mathbb{1}^{G,can,gen}}$ e $imp\_mult_{\mathbb{1}^{G,can,gen}}$ coinciden con las del objeto $\mathbb{1}^{G,can}$.
- La función $imp\_dffr_{\mathbb{1}^{G,can,gen}}\colon \mathbb{1}^{G,can,gen}_{imp_{\mathtt{CC}}} \times I^G_\sim \to I^G_\sim$ es total y definida por $imp\_dffr_{\mathbb{1}^{G,can,gen}}(df, x) := \widetilde{df}(x)$.

La relación entre los tres objetos funcionales definidos queda recogida en el siguiente teorema.

Teorema 2.9.6 *$\mathbb{1}^G$, $\mathbb{1}^{G,can}$ y $\mathbb{1}^{G,can,gen}$ son álgebras isomorfas en la categoría $\mathcal{C}^{G,\{\}}(\mathtt{CC}_{imp})$.*

El objeto $\mathbb{1}^{G,can,gen}$ es el más cercano de los tres al modo en que en EAT se representan los complejos de cadenas. Se puede decir que es un modelo teórico adecuado para representar la forma en que en EAT son manejadas las familias de complejos de cadenas.

A lo largo de los ejemplos hemos ilustrado algunas cuestiones que merece la pena destacar y que mencionamos a continuación.

2.9.4 Algunas conclusiones

Hacemos hincapié en algunos aspectos que han surgido de las aplicaciones de la teoría en ejemplos concretos y que nos parecen especialmente reseñables.

Como hemos visto, además de la presentación funcional general del objeto final, hay casos en los que pueden existir otras presentaciones, también funcionales, más reducidas.

Concretamente, hemos visto dos situaciones:

- En terminología de especificaciones ocultas, el caso en el que existen operaciones ocultas (en nuestros ejemplos aplicado a Σ_{imp} y considerando imp_Σ como único género oculto) que tienen el mismo comportamiento sobre cualquier álgebra de una categoría de álgebras ocultas (en nuestro caso, sobre la categoría $\mathcal{C}^D(\Sigma_{imp})$ habiendo fijado un G-conjunto D). En este caso, esas operaciones pueden considerarse visibles. Esto hace que pueda darse una descripción funcional del objeto final eliminando los campos funcionales correspondientes a esas operaciones. Describimos lo anterior formalmente.

  A partir de un TAD $\mathcal{T}$, para cada G-conjunto fijo D consideramos el TAD $\mathcal{T}^D = \langle \Sigma, \mathcal{C}^D(\Sigma)\rangle$. Aplicando la operación $()_{imp}$ a $\mathcal{T}^D$, obtenemos el TAD $\mathcal{T}^D_{imp} = \langle \Sigma_{imp}, \mathcal{C}^D(\Sigma_{imp})\rangle$ que modela a las familias de álgebras de $\mathcal{T}$ con conjuntos soporte D. En la categoría $\mathcal{C}^D(\Sigma_{imp})$ hemos destacado la existencia de un objeto especial, objeto que hemos denotado por $\mathbb{1}_{\mathcal{C}^D(\Sigma)}$ que, en cierto sentido, recoge a todas las álgebras de $\mathcal{C}^D(\Sigma)$. El objeto $\mathbb{1}_{\mathcal{C}^D(\Sigma)}$ es final en $\mathcal{C}^{D,\{\}}(\Sigma_{imp})$. La descripción que hemos dado para este objeto es como un espacio funcional cuyos elementos son tuplas de funciones, una por cada operación de Σ.

  En determinadas condiciones, este objeto posee descripciones más sencillas. Concretamente, si existe algún operador imp_σ cuya interpretación para cualquier álgebra de la categoría $\mathcal{C}^D(\Sigma_{imp})$ sea la misma y no dependa del género imp_Σ, la componente correspondiente a la operación σ puede suprimirse en la descripción del objeto $\mathbb{1}_{\mathcal{C}^D(\Sigma)}$.

  Así, consideramos el subconjunto Δ de Ω formado por aquellas operaciones $\sigma \in \Omega$ para las que existe una función $\tilde{\sigma}$ tal que para toda Σ_{imp}-álgebra A de $\mathcal{C}^{D,\{\}}(\Sigma_{imp})$ y para todo $x \in A_{imp_\Sigma}$, la operación $imp\_\sigma_A(x,-) = \tilde{\sigma}$. El conjunto Δ está formado por aquellas operaciones σ tales que su correspondiente operación imp_σ puede considerarse visible en Σ_{imp}. De este modo tenemos fijo un conjunto de funciones $\{\tilde{\delta}\}_{\delta\in\Delta}$.

  En esta situación, podemos definir el siguiente objeto de la categoría $\mathcal{C}^{D,\{\}}(\Sigma_{imp})$ que denotamos por $\mathbb{1}^{can}_{\mathcal{C}^D(\Sigma)}$:

  - Para cada $g \in G$, el conjunto soporte es D_g.
  - El conjunto soporte para el género distinguido imp_Σ viene dado por:

$$\mathbb{1}^{can}_{\mathcal{C}^D(\Sigma)_{imp_\Sigma}} = \Big\{(f_\delta)_{\delta\in\Omega\setminus\Delta} \mid \big\langle D, \{(f_\delta)_{\delta\in\Omega\setminus\Delta}, (\tilde{\delta})_{\delta\in\Delta}\}\big\rangle \in Obj(\mathcal{C}^D(\Sigma))\Big\}$$

  - Para cada $(\sigma\colon \omega \to v) \in \Delta$, la función $imp\_\sigma_{\mathbb{1}^{can}_{\mathcal{C}^D(\Sigma)}} : \mathbb{1}^{can}_{\mathcal{C}^D(\Sigma)_{imp_\Sigma}} \times D_\omega \to D_v$ tiene

como dominio el conjunto $\mathbb{1}^{can}_{\mathcal{C}^D(\Sigma)_{imp_\Sigma}} \times Def(\tilde{\sigma})$ y está definida por

$$imp\_\sigma_{\mathbb{1}^{can}_{\mathcal{C}^D(\Sigma)}}\left((f_\delta)_{\delta\in\Omega\setminus\Delta}, d_\omega\right) := \tilde{\sigma}(d_\omega)$$

- Para cada $(\sigma\colon \omega \to v) \in \Omega \setminus \Delta$, la función $imp\_\sigma_{\mathbb{1}^{can}_{\mathcal{C}^D(\Sigma)}} : \mathbb{1}^{can}_{\mathcal{C}^D(\Sigma)_{imp_\Sigma}} \times D_\omega \to D_v$ tiene como dominio $\left\{((f_\delta)_{\delta\in\Omega\setminus\Delta}, d_\omega) \mid d_\omega \in Def(f_\sigma)\right\}$ y está definida por

$$imp\_\sigma_{\mathbb{1}^{can}_{\mathcal{C}^D(\Sigma)}}\left((f_\delta)_{\delta\in\Omega\setminus\Delta}, d_\omega\right) := f_\sigma(d_\omega)$$

El objeto $\mathbb{1}^{can}_{\mathcal{C}^D(\Sigma)}$ pertenece a la categoría $\mathcal{C}^{D,\{\}}(\Sigma_{imp})$ y posee la propiedad recogida en el siguiente teorema.

Teorema 2.9.7 *$\mathbb{1}^{can}_{\mathcal{C}^D(\Sigma)}$ es isomorfo a $\mathbb{1}_{\mathcal{C}^D(\Sigma)}$ en $\mathcal{C}^{D,\{\}}(\Sigma_{imp})$.*

- La otra situación que hemos recogido es aquella en la que es posible determinar para todas las álgebras que se están considerando, el comportamiento de un determinado operador a partir de otro operador, digamos más sencillo. En lo anterior subyace una idea de generabilidad en un sentido amplio, que puede entenderse de diferentes formas. Por ejemplo, en el caso de los complejos de cadenas, la noción de generabilidad proviene de las propiedades matemáticas de las álgebras con las que se está trabajando (ya que éstas son complejos de cadenas libremente generados). En el caso de los conjuntos simpliciales, la situación es distinta. La noción de generabilidad subyacente está relacionada con la forma de representar los símplices de los conjuntos simpliciales, es decir, con el patrón que hemos fijado para representarlos. Al tomar como representación los símplices abstractos, lo que se obtiene es una forma universal de extender los operadores definidos solo sobre símplices geométricos a los símplices abstractos.

En los ejemplos anteriores hemos llevado a cabo las dos simplificaciones. En ambos, primero hemos reducido el número de funciones necesarias para recuperar un álgebra y después hemos visto cómo definir las funciones restantes a partir de otras. Lo anterior no significa en absoluto que siempre deba hacerse en este orden. De hecho, en los ejemplos tratados aquí es posible invertir el orden de aplicación de las simplificaciones. En el caso de los conjuntos simpliciales, también hay una forma canónica de definir los operadores de degeneración sobre los símplices abstractos, conociendo su definición sobre los símplices geométricos (el método se apoya en las propiedades de los conjuntos simpliciales que permiten intercambiar el orden de aplicación de los operadores). En el caso de los complejos de cadenas, es todavía más claro. Las operaciones de los $\mathbb{Z}$-módulos que lo forman, operaciones que hemos interpretado como visibles, también pueden definirse únicamente sobre generadores y extenderse por linealidad a todas las combinaciones. En este caso, el patrón que siguen todas las combinaciones no solo proviene de fijar una representación para los elementos de los $\mathbb{Z}$-módulos de los complejos de cadenas, sino de la propia definición de complejo de cadenas.

Como última conclusión comentar que, tal y como ha quedado patente en los ejemplos anteriores, las condiciones impuestas a las álgebras en el marco en el que hemos trabajado en este capítulo, son muy restrictivas. A base de modificar e imponer algunas restricciones adicionales a lo que pretendemos modelar, hemos conseguido adaptar casos necesarios para nosotros. Aún así, hay casos que siguen sin quedar recogidos en el marco en el que estamos trabajando. Concretamente, hay familias de Σ-álgebras que no pueden agruparse en una única

Σ_{imp}-álgebra. El tener que fijar un dominio de definición común para las operaciones de todas las álgebras de cada familia representada es muy restrictivo. Una forma de relajarlo es poder considerar como significativos solo algunos datos de ese dominio ya fijado y esto, como ya hemos comentado, es un problema de *representación* que abordaremos en el siguiente capítulo.

Sin embargo, una conclusión positiva es que, la forma más general de trabajar con familias de álgebras, algo necesario cuando durante un proceso de cálculo debe poder construirse un álgebra, es la presentación funcional de las álgebras (fijados los conjuntos subyacentes, para determinar un álgebra solo es necesario conocer cómo están definidas las operaciones). Además, las simplificaciones anteriores tienen su interpretación práctica. En lo que respecta a la primera, si una operación está definida de igual forma sobre todas las álgebras que pueden aparecer en el cálculo, no es una información necesaria para recuperar un álgebra concreta. Respecto de la segunda, si existe un modo común para todas las álgebras de definir una operación a partir de un operador que trabaja sobre un conjunto más pequeño, solo es necesario este segundo para recuperar cada álgebra concreta.

Capítulo 3

Implementación de TADs y estructuras de datos en EAT

3.1 Introducción

En el capítulo anterior hemos tratado el problema de la especificación de tipos con los que se pretende representar a familias de estructuras algebraicas (familias de grupos, de anillos, etc.). Como hemos visto, la operación $()_{imp}$ modela el paso desde un tipo de datos hasta el tipo de las familias del tipo de partida. Además esta construcción responde al patrón que siguen buena parte de las estructuras de datos que aparecen en el sistema EAT. En el presente capítulo nos vamos a centrar en el análisis formal del modo de implementación usado en EAT que, como veremos, resulta ser, en cierto sentido, canónico.

Para entender el tipo de estructuras que aparecen en dicho sistema hay que conocer que los algoritmos de cálculo que implementa, basados en métodos procedentes de la Topología Algebraica, requieren la manipulación de una gran diversidad de estructuras algebraicas, algunas de ellas complejas: grupos, grupos graduados, complejos de cadenas, conjuntos simpliciales, etc. Sin embargo, la característica principal de este software no reside tanto en la variedad de estructuras algebraicas que maneja sino en la necesidad de construir en tiempo de ejecución ejemplares de esas estructuras. Por ejemplo, construir un complejo de cadenas como paso intermedio en el cálculo de la homología de un espacio topológico (en la breve presentación de EAT que hemos incluido al principio del primer capítulo, pueden verse ejemplos de ello, concretamente en la página 6). Es en este hecho en el que radica la necesidad de manipular estructuras cuyos datos sean también estructuras (ver, por ejemplo, en la página 10 la figura 1.3, que muestra la estructura del objeto `l2s3eh` que contiene en algunos de sus campos otras estructuras como complejos de cadenas). Por tratarse de un software ya construido, y utilizado con éxito, no es nuestro interés el analizar si hubiera sido más adecuado utilizar otras estructuras de datos o implementaciones diferentes de las que fueron usadas en la creación de EAT. Por contra, nos interesa el estudio desde un punto de vista formal del modo en que realmente se hicieron las cosas en EAT.

Nuestro deseo por mantenernos tan próximos como sea posible al sistema EAT nos ha llevado a tomar algunas decisiones metodológicas de cierta importancia, que pueden resultar chocantes, y que pensamos merecen alguna explicación. Por un lado, vamos a manejar una noción de implementación que reposa en la definición de representación introducida por Hoare

en [58] (noción en la que se apoyan muchos autores que trabajan en marcos cercanos a la programación real; véanse, por ejemplo, [121] y [54]).

Así, una implementación va a consistir básicamente en una función de abstracción que relaciona datos concretos con datos abstractos (del modelo). Hemos preferido seguir esta línea, frente a la que se denomina "implementación algebraica" (seguida, por ejemplo en [39], [56], [57], [108] y [11]), por ser la más cercana a la programación concreta. En la literatura aparecen numerosos trabajos que presentan diferentes enfoques de la noción de implementación (véanse, por ejemplo, [50], [49], [54], [39], [111], [20] y [108]), en [85] se presentan y comparan algunos de ellos. Por otro lado, vamos a utilizar un lenguaje de programación concreto como lenguaje de implementación, a diferencia de lo que suele ser habitual en este tipo de estudios formales, que es utilizar un lenguaje "ad hoc" pequeño. En nuestro caso, el lenguaje elegido es Common Lisp, lenguaje en el que EAT fue implementado. Esto nos va a permitir trabajar con una sintaxis bien definida y conocida, la proporcionada por el propio lenguaje, y nos evita definir la de un supuesto lenguaje "ad hoc". Es bien conocido que no existe en la literatura una semántica formal completa de Common Lisp. Sin embargo, sí existen descripciones suficientemente precisas para nuestros fines, como las que pueden encontrarse en [88] y en [116]. La no existencia de semántica fomal nos obligará a determinar con precisión las características concretas del lenguaje que son usadas en las demostraciones de nuestros resultados, características que deberá poseer cualquier implementación correcta de Common Lisp.

Por último, no vamos a usar todo el lenguaje Common Lisp, sino que nos vamos a restringir a un subconjunto del lenguaje que nos permitirá llevar a cabo nuestro análisis de las estructuras de datos de EAT, y nos liberará de algunos problemas técnicos que pudiéramos encontrar utilizando todas las posibilidades de Common Lisp. Nuestro entorno viene dado por las siguientes condiciones:

- nos limitamos a trabajar sobre un subconjunto de Common Lisp puramente funcional: no permitimos efectos laterales (de cara al análisis de EAT esto no supone ningún problema ya que en EAT los efectos laterales aparecen solo sobre tipos elementales y no sobre el tipo de estructuras que pretendemos analizar);
- no usaremos la declaración de tipos en los códigos de los programas utilizados;
- forzamos a que todas las funciones devuelvan un único valor (que por el hecho de poder ser estructurado no supone una verdadera restricción); y,
- no consideramos los elementos propios de CLOS (Common Lisp Object System) puesto que en EAT no se requiere el uso de esta extensión de Common Lisp. Existe un software sucesor de EAT llamado Kenzo [36] en el que la programación orientada a objetos explícita juega un papel fundamental y que utiliza CLOS.

En el desarrollo del trabajo vamos a necesitar conceptos utilizados habitualmente en programación como son: tipo de datos, código, programa, etc. Algunas de estas nociones no siempre son tratadas en la literatura con todo el rigor que nosotros necesitamos. Por ello, en las primeras secciones del capítulo nos ocupamos de formalizar todos estos conceptos básicos que, pese a estar presentes en la mente de todo el que trabaja en programación, pueden dar lugar, a distintas interpretaciones y a ambigüedades. Las secciones 3.4 y 3.5 del capítulo tratan sobre las nociones de representación y de implementación. En la sección 3.6 estudiaremos la propiedad de decidibilidad de las representaciones. En la sección 3.7 analizaremos algunas implementaciones íntimamente relacionadas con las utilizadas en EAT y demostraremos que

poseen buenas propiedades desde el punto de vista de la Teoría de Categorías. El capítulo termina con aplicaciones de lo anterior a casos concretos y cercanos a EAT, exactamente se continúa con los dos últimos casos desarrollados en la parte algebraica (capítulo anterior). Al igual que en el capítulo anterior, estos ejemplos concretos nos permitirán ir un poco más lejos de lo mostrado en las secciones previas. Recogemos en el último apartado algunas conclusiones obtenidas a partir de ellos.

En [67] fueron publicados algunas de las nociones y resultados desarrollados a lo largo del capítulo.

3.2 Tipos de datos en Common Lisp

El objetivo último al programar es disponer de algún objeto informático, código, que simule el comportamiento de una determinada función. Este código tiene un dominio implícito que está compuesto por aquellos objetos (expresiones Lisp) sobre los que puede aplicarse el código terminando la evaluación y devolviendo un valor. Es habitual que este dominio implícito no coincida exactamente con el que necesitamos para simular la función, lo que suele suceder es que lo contiene. Por ejemplo, si pensamos en programar la función sucesor de un natural, podemos definir el siguiente código

$$\mathtt{c_s} \quad \equiv \quad \begin{array}{l}\texttt{\#'(lambda (x)}\\ \qquad\qquad \texttt{(+ 1 x))}\end{array}$$

cuyo dominio implícito está formado por todos aquellos objetos Lisp a los que se les puede sumar uno que, evidentemente, no coincide con $\mathbb{N}$ que es el que pretendemos representar. Esto hace que sea natural asociar a un código un conjunto de objetos sobre los que considerar la evaluación del código. Ligada a esta necesidad, aparece una noción básica en programación, la noción de tipo de datos, sobre la que existen múltiples variantes en la literatura.

En este apartado vamos a tratar la noción de tipo de datos en Common Lisp, apoyándonos en la descripción detallada de este lenguaje que puede encontrarse en [116]. En particular, la noción de tipo de datos y las directamente relacionadas con ella se desarrollan entre los capítulos 2 y 6 de [116].

Como punto de partida consideramos el Universo de datos formado por todos los objetos Common Lisp, al que de ahora en adelante nos referiremos por $\mathcal{U}$. Sin embargo, lo habitual es que al programar algo concreto trabajemos sobre un subconjunto de objetos del universo del lenguaje. Precisamente eso es lo que en Common Lisp se denomina tipo de datos (véase [116], página 12).

Definición 3.2.1 *Un* tipo de datos *es cualquier conjunto de objetos Lisp.*

Los tipos de datos en Common Lisp se organizan en una jerarquía que es un orden parcial respecto a la relación de contenido. Así, un objeto Lisp no tiene tipo sino que puede pertenecer o no a un tipo. Se puede afirmar que Common Lisp no es un lenguaje fuertemente tipado. Pese a ello, posee suficientes herramientas para actuar como lenguaje tipado, en particular, permite declaración de tipos, aunque esta faceta no va a ser tratada en esta memoria.

La noción de subtipo se define de modo natural: dados dos tipos de datos $\mathtt{T}_1$ y $\mathtt{T}_2$ se dice que $\mathtt{T}_1$ es *subtipo de* $\mathtt{T}_2$ si todo objeto Lisp que pertenece a $\mathtt{T}_1$ también pertenece a $\mathtt{T}_2$.

Al igual que en cualquier lenguaje, existen conjuntos de objetos Lisp de interés especial

que se denominan *tipos de datos predefinidos* y que el propio lenguaje nomina por medio de determinados símbolos estándar. Una enumeración exhaustiva de éstos se encuentra en [116], página 50. Algunos tipos de datos pueden nominarse y a los objetos Lisp que los nominan se les llama *especificadores de tipos*. Los especificadores de tipos pueden ser símbolos, que nominan a los tipos de datos predefinidos, o listas, caso en el que nominan a combinaciones o especializaciones de otros tipos de datos (el capítulo 4 de [116] los recoge). En Common Lisp, el tipo de datos formado por todos los objetos Lisp se especifica por el símbolo `t`, es decir, `t` representa al Universo $\mathcal{U}$ de todos los objetos Lisp y, cualquier otro tipo de datos es subtipo de `t`. En el extremo opuesto, el símbolo `nil` especifica al tipo de datos vacío.

En cualquier lenguaje es de gran importancia conocer cómo se trata la *igualdad* entre objetos. En este sentido, Common Lisp proporciona un abanico de predicados de igualdad que, ordenados de menor a mayor generalidad, son: `eq`, `eql`, `equal` y `equalp`; además de ellos, `=` que actúa sobre los datos numéricos (su comportamiento puede consultarse en [116], páginas 103 y siguientes). Cada tipo de datos predefinido tiene asociado, como más específico, uno de los predicados de igualdad anteriores. Del mismo modo, los tipos definidos como especialización o combinación de otros poseen también una igualdad específica que viene inducida por la de los tipos a partir de los que se han definido. Sin embargo, al trabajar con tipos de datos es frecuente que la igualdad que se necesite no coincida con la dada por el predicado que el tipo tiene asociado, ni tampoco con ninguna de las definidas mediante los predicados de igualdad proporcionados por el lenguaje. En estos casos, el programador debe definir su propia igualdad, que se ajuste al contexto en el que está trabajando. Así, normalmente no se trabaja solo con tipos sino que el punto de partida es un tipo de datos junto con una relación de equivalencia definida sobre los objetos del tipo (es decir, un conjunto con igualdad; ver definición en preliminares, página 15), relación que no necesariamente tendrá que venir descrita por una expresión algorítmica. Esto motiva la siguiente definición.

Definición 3.2.2 *Un* dominio Common Lisp *es un tipo de datos junto con una igualdad, dada por una relación de equivalencia definida sobre los datos pertenecientes al tipo. Denotamos un dominio Common Lisp por un par* $(\mathtt{T}, =_{\mathtt{T}})$.

Como hemos dicho, cada tipo de datos `T` dispone de una igualdad específica dada por el propio lenguaje Common Lisp. Al correspondiente dominio lo denotamos por $(\mathtt{T}, =_{\mathtt{CL}})$. Como ejemplo, `eq` es la igualdad específica sobre el tipo `symbol`, sobre tipos de datos formados por objetos Lisp numéricos la igualdad específica es la dada por $=$, sobre tipos de datos estructurados como listas, vectores, registros, etc. se define recursivamente a partir de las anteriores. Siempre que no haya lugar a confusión hablaremos del dominio `T` para referirnos realmente al dominio $(\mathtt{T}, =_{\mathtt{CL}})$. En la definición anterior está implícito el hecho de que $=_{\mathtt{T}}$ debe ser compatible con $=_{\mathtt{CL}}$ que es la que posee `T` como conjunto.

Al trabajar en un universo $\mathcal{U}$ y considerar subconjuntos de él aparece de modo natural la noción de decidibilidad. Fijado un universo, un conjunto de objetos del universo es *decidible* si existe un algoritmo que determina si un objeto del universo pertenece o no al conjunto. En principio, la noción general de decidibilidad no coincide con la decidibilidad en Common Lisp. La noción de decidibilidad en Common Lisp viene dada por el predicado predefinido `typep` que determina si un objeto Lisp pertenece o no a un determinado tipo de datos, `(typep object type)`. Los argumentos de `typep` son un objeto Lisp y un especificador de tipo y el predicado comprueba si el objeto pertenece o no al tipo nominado por el especificador.

Definición 3.2.3 *Un tipo de datos Common Lisp es* decidible Common Lisp *si posee un espe-*

cificador de tipo admitido por el predicado `typep`.

La idea general de decidibilidad coincide con la noción de tipo discriminable en Common Lisp. Un *tipo de datos discriminable Common Lisp* es aquel para el que existe un predicado que decide si un objeto Lisp cualquiera pertenece o no al tipo, es decir, es discriminable si el tipo es decidible en el universo de objetos Common Lisp. Teniendo en cuenta que los especificadores de tipos admitidos por el predicado `typep` son aquellos que pueden usarse como discriminadores (incluidos los definidos utilizando el predicado `satisfies`), es claro que un tipo de datos es decidible Common Lisp si y solo si es discriminable Common Lisp. Como observación comentar que el único tipo Common Lisp para el que los fines de declaración y discriminación son distintos es la especialización del tipo `function`: `(function (arg1-type arg2-type ... ) value-type)`, tipo que solamente puede usarse con fines de declaración pero no de discriminación, por lo que el tipo de datos cuyo especificador es `(function (arg1-type arg2-type ... ) value-type)` no es un tipo decidible Common Lisp. La noción de *decidibilidad Common Lisp* coincide con la noción general de *decidibilidad*, en este caso en el universo de objetos Lisp.

Definición 3.2.4 *Se dice que un dominio* $(\mathtt{T}, =_{\mathtt{T}})$ *es decidible Common Lisp si lo es el tipo* `T`.

Observar que en la definición anterior no se dice nada respecto de la decidibilidad de $=_{\mathtt{T}}$. De hecho, las igualdades específicas sobre cada tipo de datos proporcionadas por el propio lenguaje son decidibles, mientras que las definidas por los usuarios pueden serlo o no.

La forma habitual en que se plantea el problema de la decidibilidad es la existencia o no de un test de pertenencia (que es un algoritmo) de los elementos de un subconjunto. Si A es un conjunto en un universo y B es subconjunto de A, se dice que B es *decidible en* A si existe un algoritmo que comprueba la pertenencia a B de todos los elementos de A. Extendemos al marco de los tipos de datos Common Lisp la noción anterior.

Definición 3.2.5 *Sea* `A` *un tipo Common Lisp y sea* `B` *un subtipo de* `A`. *Se dice que* `B` *es* decidible en `A` *si existe un predicado Lisp que determina si un objeto de* `A` *está o no en* `B`.

Si un tipo `A` es decidible Common Lisp y un subtipo suyo `B` también lo es, el predicado que comprueba la pertenencia de los objetos de $\mathcal{U}$ a `B` sirve para comprobar la pertenencia a `A`, por lo que, por definición, `B` también es decidible en `A`. Pero además, también se cumple el recíproco. De hecho, la función de test necesaria para probar la decidibilidad Common Lisp de `B` podemos escribirla como sigue:

```
(defun decidibleBenU (x)
     (and (decidibleAenU x)
          (decidibleBenA x)))
```

siendo `decidibleAenU` la función de test de `A` en $\mathcal{U}$ y `decidibleBenA` la de `B` en `A`.

Como observación, indicar que teniendo en cuenta que la definición de decidibilidad Common Lisp en términos del predicado `typep` coincide con la de decidibilidad en general, cabría esperar que la noción de subtipo decidible de un tipo decidible pudiera darse en términos de la primitiva `subtypep`. Sin embargo, esta primitiva no sirve para saber si un conjunto de objetos Lisp es o no subtipo de un tipo decidible Common Lisp, ya que sobre especificadores de tipos que utilizan determinadas primitivas como `satisfies`, `and`, `or`, etc., no es capaz de determinar la relación entre los dos tipos nominados por los especificadores que aparecen como argumentos (véase [116], página 97). Por ejemplo, `subtypep` no es capaz de decidir si el tipo `integer` es

o no subtipo del tipo especificado por `(satisfies integerp)`, ni tampoco si la relación es la inversa.

Ahora ya tenemos las bases para introducir los conjuntos de objetos Lisp con los que nos interesa trabajar, los decidibles.

Definición 3.2.6 *Un* Tipo Concreto Common Lisp (TCCL) *es un dominio decidible Common Lisp.*

Según la definición anterior, un tipo concreto Common Lisp puede ser:

- un tipo predefinido Common Lisp: `integer, symbol, ...` ;
- un tipo construido a partir de otro TCCL utilizando los constructores de tipos que Common Lisp proporciona: `struct, array, list, ...` ; o
- un subconjunto decidible de los anteriores, es decir, un subconjunto para el que exista un predicado Lisp capaz de decidir si un objeto de un tipo está o no en el subconjunto.

En lo que sigue, al hablar de tipo o de tipo Common Lisp nos referimos a un TCCL.

La noción de *expresión correcta* en Common Lisp también requiere algunas explicaciones. En un lenguaje fuertemente tipado el concepto de expresión bien construida se identifica con el de expresión *sintácticamente correcta*. Sin embargo, esto no ocurre en lenguajes como Common Lisp, con tipado implícito, dinámico, en los que una expresión puede estar o no bien construida dependiendo del entorno en el que se considere. Hay expresiones, como por ejemplo, `(integerp x x)` que, independientemente del entorno no están bien construidas, y expresiones como `(if (integerp x)(+ x x))` construidas correctamente para cualquier entorno. Sin embargo, una expresión como `(+ x x)` puede estar bien o mal construida, depende del tipo al que pertenezca `x` en el momento de la evaluación. Por ejemplo, si `x` pertenece al tipo `integer` la expresión anterior estará bien construida mientras que si `x` pertenece al tipo `symbol` la expresión no lo estará. Este ejemplo muestra que no es conveniente identificar la noción de expresión bien construida con la de corrección sintáctica, siendo más adecuado considerar otros tipos de corrección. Diremos que una expresión es *estructuralmente correcta* si está correctamente construida para algún tipo de datos. Así, las dos expresiones Common Lisp anteriores `(if (integerp x)(+ x x))` y `(+ x x)` son estructuralmente correctas. Los ejemplos anteriores nos muestran también que en el estudio de la corrección estructural de una expresión hay, al menos, dos factores a tener en cuenta. Uno directamente relacionado con la sintaxis de la expresión y que tiene que ver con el número de argumentos que puede tener la expresión (número que depende de la operación a evaluar) y, otro, relacionado con el tipo que pueden tener los argumentos sobre los que aplicar la función.

La noción de corrección estructural está directamente relacionada con la *inferencia de tipos* en lenguajes como ML (debida a Milner [80]). La inferencia de tipos consiste en asociar un *dominio* a cada expresión, dominio que estará formado por todos aquellos objetos sobre los que puede evaluarse la expresión. Por ejemplo, si observamos la expresión `(+ x (first x))` vemos que para poder aplicar la primitiva `first` a `x`, `x` debe pertenecer al tipo `list` (véase [116], página 415). Además, para aplicar la primitiva `+` a `x` es necesario que `x` satisfaga el predicado `numberp`. Sin embargo, ambos tipos, `list` y `number` son disjuntos (véase [116], páginas 38 y 39), por lo que, no existe ningún objeto Lisp sobre el que la expresión anterior pueda evaluarse. Esto prueba que la expresión anterior es estructuralmente incorrecta. Así, la corrección estructural

de una expresión está relacionada con su dominio implícito, que puede inferirse a partir de ella. Según esto podemos decir que una expresión es *estructuralmente correcta* si el dominio implícito no es vacío.

Las expresiones estructuralmente correctas son una de las bases sobre las que reposa la noción de programa. Intuitivamente, las expresiones estructuralmente correctas forman parte del *código* del programa y habrá que tener en cuenta los dominios implícitos de éstas antes de decidir sobre qué datos evaluarlas.

3.3 Programas en Common Lisp

Uno de los conceptos básicos en programación es el de *programa*. Es habitual encontrar trabajos en los que, implícita o explícitamente, se identifica programa con código, sin embargo, esta identificación da lugar a una noción de programa muy pobre que no sirve para recoger la noción de función. Intuitivamente, un programa va a ser un código que al aplicarlo sobre determinados objetos debe comportarse de cierta manera. En la afirmación anterior hay cuestiones que concretar, por ejemplo, ¿sobre qué datos el programa no debe producir errores en las llamadas? ¿sobre cuáles debe terminar?; si termina ¿debe devolver algo? ¿qué puede decirse del dato que devuelve? El objetivo de esta sección es concretar la noción de programa, dar una definición formal que elimine toda esta ambigüedad. Un tipo que será de especial relevancia en esta tarea es el tipo de los objetos funcionales. Intuitivamente, un objeto funcional corresponde con lo que se considera habitualmente como el código de un programa. El predicado predefinido Common Lisp `functionp` sirve para determinar el dominio del tipo de los objetos funcionales.

Definición 3.3.1 *Un objeto funcional es un objeto Lisp que satisface el predicado* `functionp`.

Dicho de otro modo, un objeto funcional es un objeto Lisp que al aplicarle la primitiva `functionp` no devuelve `nil`. (En Common Lisp cualquier objeto distinto de `nil` se interpreta como `t`.) Por construcción, `functionp` devuelve un objeto distinto de `nil` si su argumento puede aplicarse a otros objetos Lisp por medio de las funciones `funcall` o `apply` (véase [116], página 75). Según la definición anterior, los objetos funcionales son los definidos con `defun` y los cierres léxicos de expresiones lambda. (Los cierres léxicos, ver en [116] páginas 145 y 148, y las expresiones lambda, ver en [116] páginas 75–83, son objetos Lisp con propiedades especiales y que en nuestro trabajo aparecen de forma esencial. Los cierres léxicos están relacionados con el alcance de los objetos Lisp. La idea general es que ambos juntos, es decir, cierres léxicos de expresiones lambda, permiten simular la programación orientada a objetos, debido a que cada cierre léxico de una expresión lambda se lleva consigo el contexto en ese momento, es decir, se lleva consigo el estado total en el que está el intérprete de Common Lisp. Esto lo iremos viendo a lo largo del capítulo en el que habrá puntos concretos en los que haremos uso de ello.) La igualdad específica sobre el tipo de los objetos funcionales, es decir, la igualdad que hemos denotado por $=_{\texttt{CL}}$, es la dada por `eq`.

Toda expresión de la forma `(funcall x a1 ... an)` donde `x` es un objeto funcional y `a1`, ... , `an` son objetos cualesquiera es estructuralmente correcta. Así, si consideramos el siguiente objeto funcional

```
(defun SumaUno (x)
     (+ 1 x))
```

las llamadas `(funcall #'SumaUno x)` y `(funcall #'SumaUno x x)` son estructuralmente correctas. Sin embargo, la evaluación de cada una de ellas produce distintos efectos, que vienen determinados por el comportamiento establecido en Common Lisp para el tratamiento de "situaciones excepcionales".

Definición 3.3.2 *En Common Lisp, una* situación *es la evaluación de una expresión en un contexto específico.*

La propia descripción de un objeto funcional permite conocer su comportamiento en "situaciones normales", es decir, en evaluaciones de llamadas al objeto que terminan y devuelven un objeto Lisp. Por ejemplo, la evaluación de la llamada `(funcall #'SumaUno x)` termina y devuelve un objeto Lisp para cualquier `x` objeto Lisp que satisfaga el predicado `numberp` (véase la página 100 de [116]). Sin embargo, también podemos encontrar "situaciones excepcionales", es decir, evaluaciones de expresiones cuyo efecto no está claro. Un ejemplo de una situación excepcional es evaluar la llamada anterior `(funcall #'SumaUno x)` sobre cualquier objeto Lisp `x` que no satisfaga el predicado `numberp`. Para analizar las situaciones excepcionales que pueden producirse durante la evaluación de una llamada a un objeto funcional, recurrimos de nuevo a [116], concretamente a su capítulo 29, que incorpora un *Sistema de Condiciones Common Lisp* desarrollado para la detección y manipulación de las situaciones excepcionales. (Realmente para nuestro análisis sería suficiente con recurrir al capítulo 24 de esa misma referencia, capítulo ya incorporado en la versión anterior [115] donde se analizan las situaciones de error pero no se permite su manipulación. Pese a que no vamos a usar nada sobre la manipulación de errores, como Common Lisp estándar incorpora ya el sistema de condiciones, nos ha parecido más interesante basarnos en él en este punto que limitarnos simplemente a lo incluido en la primera versión.) Las siguientes definiciones corresponden a las nociones de condición y de error.

Definición 3.3.3

i) *Una* condición en Common Lisp *es una situación "interesante" en un programa que se ha detectado y comunicado. La acción por la que un programa muestra que se ha detectado una condición se denomina* señalar.

ii) *Un* error en Common Lisp *es una condición en la que la ejecución normal de un programa no puede continuar sin una intervención externa. (La intervención externa puede provenir del usuario o de algún programa de control diseñado para ello con las herramientas que el propio lenguaje proporciona.)*

iii) *En Common Lisp, que un programa* señale un error *significa que el programa reconoce que no sabe cómo continuar la ejecución y que necesita intervención externa para continuar.*

La primitiva Common Lisp para señalar condiciones es la función `signal`, a partir de la cual se construyen otras. En particular, la forma más simple en la que cualquier implementación de Common Lisp que siga fielmente la definición del lenguaje señala un error es por medio de la función `error`, a través de la interfaz de usuario y enviando un mensaje determinado. Es habitual encontrar implementaciones en las que el comportamiento varía porque se han programado otras utilidades, pero nosotros nos limitamos a utilizar lo descrito en la definición formal del lenguaje. No nos planteamos un estudio exhaustivo de las condiciones Common Lisp, ni tan siquiera de los errores, sino que vamos simplemente a intentar detectar y caracterizar aquellas condiciones que nos interesan, que son las que pueden producirse en la evaluación

de llamadas estructuralmente correctas a objetos funcionales. Serán éstas las que influirán en nuestro objetivo más inmediato que es formalizar la noción de programa y, por tanto, las que nos interesa caracterizar. En concreto, son las siguientes:

i) llamada a un objeto funcional con un *número incorrecto de argumentos*,

ii) llamada a un objeto funcional con *argumentos de tipo incorrecto*,

iii) llamada a un objeto funcional cuya evaluación no termina, y

iv) llamada a un objeto funcional con *número y argumentos de tipo correctos* pero que señala un error.

En las páginas 916 y siguientes de [116] podemos encontrar una clasificación de los tipos predefinidos de condiciones que cualquier implementación de Common Lisp debe poseer. Vamos a utilizar algunos de ellos para caracterizar algunas de las situaciones anteriores.

Definición 3.3.4 *Un objeto funcional* `x` *se dice que* puede aplicarse sobre n argumentos *si para cualquier llamada* `(funcall x` $\mathtt{a}_1$ `...` $\mathtt{a}_n$`)`*, con* $\mathtt{a}_1, \ldots, \mathtt{a}_n$ *objetos Lisp cualesquiera, no se señala un error que envía un mensaje del tipo* "número incorrecto de argumentos".*

Definición 3.3.5 *Un objeto funcional* `x` *que puede aplicarse sobre n argumentos se dice que* puede aplicarse sobre los tipos de datos $\mathtt{T}_1, \ldots, \mathtt{T}_n$ *si para cualquier llamada* `(funcall x` $\mathtt{a}_1$ `...` $\mathtt{a}_n$ `)`*, con* $\mathtt{a}_i \in \mathtt{T}_i$ *para todo* $i = 1, \ldots, n$*, no se señala un error que envía un mensaje del tipo* "se le ha dado a la función un argumento de tipo equivocado".

Las dos definiciones cubren las situaciones recogidas en *i*) y *ii*) y muestran que considerar un objeto funcional como código de un programa no es independiente de los argumentos sobre los que se quiere trabajar.

Así, en el caso del objeto funcional `#'SumaUno`, la evaluación de la llamada `(funcall #'SumaUno x)` sobre un objeto Lisp que no satisfaga el predicado `numberp` señalará un error, error provocado porque el argumento dado en la llamada no pertenece a ningún tipo sobre el que el objeto funcional `#'SumaUno` puede trabajar. Si consideramos la llamada `(funcall #'SumaUno x x)`, sea cual sea el objeto Lisp `x`, su evaluación señalará un error porque el número de argumentos aportados para la evaluación de la llamada es incorrecto.

Para caracterizar las situaciones recogidas en el punto *iii*) anterior, es decir, las situaciones de no terminación en llamadas a objetos funcionales, es necesario introducir la siguiente definición.

Definición 3.3.6 *Sean* $\mathcal{U}_1$ *y* $\mathcal{U}_2$ *conjuntos de objetos Lisp con* $\mathcal{U}_2 \neq \emptyset$*. Un* algoritmo Common Lisp de $\mathcal{U}_1$ en $\mathcal{U}_2$ *es un objeto funcional que al invocarlo con* `funcall` *termina su evaluación para cualquier objeto Lisp tomado en* $\mathcal{U}_1$ *y devuelve un objeto de* $\mathcal{U}_2$*.*

Denotamos por $\mathcal{A}(\mathcal{U}_1, \mathcal{U}_2)$ al conjunto de algoritmos Common Lisp de $\mathcal{U}_1$ en $\mathcal{U}_2$. Según esta notación, $\mathcal{A}(\mathcal{U}, \mathcal{U})$ denota al conjunto de predicados Lisp. Observar que el conjunto de

*Obsérvese que, debido a la riqueza de Common Lisp en lo que se refiere a los parámetros (admite parámetros opcionales, parámetros con palabra clave, etc.; véase [116], páginas 75–83), hay objetos funcionales que pueden ser aplicados sobre distinto número de argumentos. Por ejemplo, una primitiva como `list` puede aplicarse sobre cualquier número de argumentos n (incluyendo el caso $n = 0$).

predicados Lisp no es decidible ya que el especificador de tipo asociado `(function (t) t)` no es admitido por `typep`.

Por ejemplo, el siguiente objeto Lisp

```
#'(lambda (  )
     (do ( )
         (nil)))
```

es un objeto funcional, pero no es un algoritmo Common Lisp sobre ningún tipo Common Lisp ya que ninguna llamada al objeto funcional anterior termina. (En el caso del objeto anterior, no se señala ningún error.)

Que un objeto funcional sea o no un algoritmo no sólo depende del propio objeto funcional, sino que también depende de los tipos $\mathcal{U}_1$ y $\mathcal{U}_2$. Por ejemplo, el objeto funcional

```
#'(lambda (n)
     (do ( )
         ((zerop n) t)
         (setq n (- n 1))))
```

admite como dominio el conjunto formado por los objetos Lisp que satisfacen el predicado

```
(defun naturalp (n)
          (and integerp (>= n 0)))
```

por lo que es un algoritmo del tipo especificado por `(satisfies naturalp)` en el tipo `t`.

La noción sobre la que descansa la definición de programa no es la de objeto funcional por sí mismo, sino la de algoritmo Common Lisp. El interés no radica en considerar los objetos funcionales aisladamente sino que deben llevar asociados los objetos sobre los que realizar las llamadas. Un mismo objeto funcional `c` puede pertenecer a distintos conjuntos $\mathcal{A}(\mathcal{U}_1,\mathcal{U}_1')$ y $\mathcal{A}(\mathcal{U}_2,\mathcal{U}_2')$ pudiendo guardar las parejas $(\mathcal{U}_1,\ \mathcal{U}_1')$ y $(\mathcal{U}_2,\ \mathcal{U}_2')$ cualquier relación de contenido entre ellas. Por ejemplo, el objeto funcional

```
#'(lambda (x)
        (+ 1 x))
```

pertenece a $\mathcal{A}$(`(satisfies numberp)`, `(satisfies numberp)`), $\mathcal{A}$(`(satisfies integerp)`, `(satisfies integerp)`), $\mathcal{A}$(`(satisfies integerp)`, `(satisfies numberp)`) y también a $\mathcal{A}$(`(satisfies numberp)`, `t`), pero no pertenece, por ejemplo, a $\mathcal{A}$(`t`, `t`).

Observar que los objetos funcionales pueden tener cualquier número de argumentos n con $n \geq 0$, y normalmente el tipo $\mathcal{U}_1$ convendrá verlo como un producto $\mathtt{T}_1 \times \ldots \times \mathtt{T}_n$. Sin embargo, el objeto devuelto siempre es uno, por lo que $\mathcal{U}_2$ lo veremos como un tipo `T`.

La noción de algoritmo Common Lisp corresponde con la de función total de las Matemáticas, para la que se tiene un dominio, un codominio y una descripción del comportamiento sobre los datos del dominio. Con la de programa pretendemos recoger también la simulación de las funciones parciales. Por ello nos interesa detectar y analizar la situación recogida en el

punto *iv*) de la página 132, es decir, recoger el caso de las llamadas a objetos funcionales con número y tipo de argumentos correctos pero que producen error. Un ejemplo de esta situación sería evaluar una llamada al siguiente objeto funcional

```
#'(lambda (n  m)
        (/ n m))
```

sobre los objetos Lisp `5` y `0`. Estas situaciones corresponden a llamadas a objetos funcionales a los que se les pueden asociar tipos que constituyan el dominio y el codominio ($\mathbb{Z} \times \mathbb{Z}$ y $\mathbb{Z}$, respectivamente, en el ejemplo), pero de forma que existen objetos del dominio sobre los que se realiza alguna operación no permitida. Las situaciones como la anterior señalan errores del tipo `arithmetic-error` (véase [116], página 917 y siguientes).

En estos comentarios nos hemos centrado en las llamadas a objetos funcionales a través de la primitiva `funcall`, pero las llamadas también pueden realizarse mediante la primitiva `apply`. Si `x` es un objeto funcional, las llamadas `(funcall x` $\mathtt{a}_1$ `...` $\mathtt{a}_n$`)` y `(apply x (list` $\mathtt{a}_1$ `...` $\mathtt{a}_n$`))` son equivalentes en el sentido de que para cualesquiera objetos Lisp $\mathtt{a}_1$ `...` $\mathtt{a}_n$ en una se señalará error si y solo si se señala en la otra, la evaluación de la primera llamada termina si y solo si lo hace la segunda y, en ese caso, el objeto devuelto por ambas es el mismo. Por ello, a partir de ahora, para trabajar con llamadas a objetos funcionales, nos vamos a limitar a trabajar con una de ellas, concretamente, utilizaremos `funcall`.

Con esto ya tenemos las herramientas suficientes para considerar funciones parciales. Por ejemplo, la función matemática $f(x) := x + 1$ sobre los naturales, podría simularse por medio del objeto funcional

```
#'(lambda (x)
        (+ 1 x))
```

con dominio y codominio el tipo `integer` y tomando como dominio real los objetos que satisfacen el predicado `naturalp` definido en la página 134. Sobre los objetos de `integer` el objeto funcional anterior puede aplicarse y sobre los que satisfacen `naturalp`, la evaluación termina y devuelve otro objeto que también lo verifica. Así que este objeto pertenece a $\mathcal{A}$(`(satisfies naturalp)`, `(satisfies naturalp)`).

Con esta idea introducimos la siguiente definición.

Definición 3.3.7 *Un* programa *(Common Lisp) es una tupla* $\mathtt{p} = (\mathtt{c}^{\mathtt{p}}, \mathtt{S}^{\mathtt{p}}, \mathtt{T}_1^{\mathtt{p}}, \ldots, \mathtt{T}_n^{\mathtt{p}}, \mathtt{T}^{\mathtt{p}})$ *donde:* $\mathtt{T}_1^{\mathtt{p}}, \ldots, \mathtt{T}_n^{\mathtt{p}}, \mathtt{T}^{\mathtt{p}}$ *son tipos concretos Common Lisp;* $\mathtt{S}^{\mathtt{p}}$ *es un subconjunto de* $\mathtt{T}_1^{\mathtt{p}} \times \ldots \times \mathtt{T}_n^{\mathtt{p}}$*; y,* $\mathtt{c}^{\mathtt{p}}$ *es un objeto funcional que puede ser aplicado sobre los tipos de datos* $\mathtt{T}_1^{\mathtt{p}}, \ldots, \mathtt{T}_n^{\mathtt{p}}$*, que es un algoritmo del conjunto* $\mathcal{A}(\mathtt{S}^{\mathtt{p}}, \mathtt{T}^{\mathtt{p}})$ *y que respeta las igualdades de los tipos concretos.*

En la definición anterior se está exigiendo que la evaluación de (`funcall` $\mathtt{c}^{\mathtt{p}}$ $\mathtt{d}_1$ `...` $\mathtt{d}_n$) termine para cualquier $(\mathtt{d}_1, \ldots, \mathtt{d}_n) \in \mathtt{S}^{\mathtt{p}}$ y devuelva un elemento de $\mathtt{T}^{\mathtt{p}}$. A este elemento lo denotaremos por $\mathtt{c}^{\mathtt{p}}(\mathtt{d}_1, \ldots, \mathtt{d}_n)$. Denominaremos al objeto funcional $\mathtt{c}^{\mathtt{p}}$ *código del programa*, al conjunto $\mathtt{S}^{\mathtt{p}}$ *soporte* o *dominio de definición del programa*, a $\mathtt{T}_1^{\mathtt{p}}, \ldots, \mathtt{T}_n^{\mathtt{p}}$ los llamaremos *tipos de entrada* y a $\mathtt{T}^{\mathtt{p}}$ *tipo de salida.*

Que el objeto funcional $\mathtt{c}^{\mathtt{p}}$ respete las igualdades de los tipos concretos significa que los objetos devueltos al aplicar el objeto funcional $\mathtt{c}^{\mathtt{p}}$ sobre objetos iguales en los dominios (por la igualdad del tipo concreto) deben ser iguales en el tipo de salida, es decir, debe verificar que

si $(\mathtt{d}_1, \ldots, \mathtt{d}_n)$ y $(\mathtt{d}'_1, \ldots, \mathtt{d}'_n)$ son elementos de $\mathtt{S^p}$ tales que $\mathtt{d}_i =_{\mathtt{T}^\mathtt{p}_i} \mathtt{d}'_i$ para todo $i = 1, \ldots, n$, entonces $\mathtt{c^p}(\mathtt{d}_1, \ldots, \mathtt{d}_n) =_{\mathtt{T^p}} \mathtt{c^p}(\mathtt{d}'_1, \ldots, \mathtt{d}'_n)$. Así, cada programa $\mathtt{p} = (\mathtt{c^p}, \mathtt{S^p}, \mathtt{T}^\mathtt{p}_1, \ldots, \mathtt{T}^\mathtt{p}_n, \mathtt{T^p})$ define una función (parcial), con dominio de definición $\mathtt{S^p}$, que a un elemento $\mathtt{s}$ de $\mathtt{S^p}$ lo aplica sobre $\mathtt{c^p}(\mathtt{s})$.

Definición 3.3.8 *Dado un programa* $\mathtt{p} = (\mathtt{c^p}, \mathtt{S^p}, \mathtt{T}^\mathtt{p}_1, \ldots, \mathtt{T}^\mathtt{p}_n, \mathtt{T^p})$ *se llama* función definida por el programa $\mathtt{p}$*, y se denota por* $F(\mathtt{p})$*, a la función parcial con dominio* $Def(F(\mathtt{p})) = \mathtt{S^p}$ *dada por* $F(\mathtt{p})(\mathtt{s}) = \mathtt{c^p}(\mathtt{s})$.

Además, por respetar las igualdades, $F(\mathtt{p})$ induce una función con dominio el cociente de $\mathtt{T}^\mathtt{p}_1 \times \ldots \times \mathtt{T}^\mathtt{p}_\mathtt{n}$ por la relación de equivalencia que definen las igualdades en los $\mathtt{T}^\mathtt{p}_i$, $i = 1, \ldots, n$.

3.4 Representaciones de TADs

Uno de los principales problemas que un programador debe resolver es cómo representar en la máquina el modelo que tiene en su mente. En [64] abordamos este problema y ahí puede encontrarse una primera aproximación al tratamiento que realizamos en esta memoria.

Los objetos de cualquier lenguaje de programación, en particular los objetos Lisp, se usan como representaciones de los datos abstractos que forman el modelo. Por ello, los objetos de los lenguajes de programación pueden ser vistos como "datos concretos" que representan en la máquina a "datos abstractos". Para relacionar datos concretos con los abstractos a los que están representando introducimos la definición de *representación*.

Definición 3.4.1 *Una* representación *es una tupla* $\mathcal{R} = (\mathcal{D}_\mathcal{R}, \mathcal{S}_\mathcal{R}, =_\mathcal{R}, \alpha_\mathcal{R}, \mathcal{M}_\mathcal{R})$ *donde* $\mathcal{D}_\mathcal{R}$ *es un tipo concreto Common Lisp llamado* dominio de la representación*;* $\mathcal{S}_\mathcal{R}$ *es un subconjunto de* $\mathcal{D}_\mathcal{R}$ *llamado* soporte de la representación*;* $=_\mathcal{R}$ *es una relación de equivalencia sobre* $\mathcal{S}_\mathcal{R}$ *(más grande que la relación de equivalencia que* $\mathcal{S}_\mathcal{R}$ *hereda del dominio* $\mathcal{D}_\mathcal{R}$*), llamada* igualdad de la representación*;* $\mathcal{M}_\mathcal{R}$ *es un conjunto llamado* modelo de la representación*; y, finalmente,* $(\alpha_\mathcal{R}, \mathcal{S}_\mathcal{R}, \mathcal{D}_\mathcal{R}, \mathcal{M}_\mathcal{R})$ *es una función parcial, llamada* función de abstracción*, compatible con la igualdad de la representación.*

La definición anterior está basada en la introducida por Hoare en [58]. La diferencia entre ambas es que la anterior está dada en términos del soporte de la representación mientras que Hoare la dio en términos de lo que se denomina *invariante de la representación*.

Definición 3.4.2 *Dada una representación* $\mathcal{R} = (\mathcal{D}_\mathcal{R}, \mathcal{S}_\mathcal{R}, =_\mathcal{R}, \alpha_\mathcal{R}, \mathcal{M}_\mathcal{R})$*, se llama* invariante de la representación *a la función característica de* $\mathcal{S}_\mathcal{R}$ *como subconjunto de* $\mathcal{D}_\mathcal{R}$.

Algunos comentarios sobre las igualdades involucradas en la definición de representación nos parecen necesarios. En la noción de representación aparecen, implícita o explícitamente, las siguientes igualdades: la igualdad que $\mathcal{S}_\mathcal{R}$ hereda como subconjunto del dominio $\mathcal{D}_\mathcal{R}$ $(=_{\mathcal{S}_\mathcal{R}})$, la de la representación $(=_\mathcal{R})$, la del modelo $(=)$ y la definida por la función de abstracción $(=_{\alpha_\mathcal{R}})$. En la definición de representación se exige que la igualdad de la representación $=_\mathcal{R}$ contenga a la del soporte $=_{\mathcal{S}_\mathcal{R}}$. Por otro lado, podemos considerar la igualdad que en $\mathcal{S}_\mathcal{R}$ define la función de abstracción $\alpha_\mathcal{R}$. Dos objetos $\mathtt{t_1}$ y $\mathtt{t_2}$ de $\mathcal{S}_\mathcal{R}$ se identifican por $=_{\alpha_\mathcal{R}}$, $\mathtt{t_1} =_{\alpha_\mathcal{R}} \mathtt{t_2}$, si sus imágenes por $\alpha_\mathcal{R}$ son iguales en $\mathcal{M}_\mathcal{R}$, $\alpha_\mathcal{R}(\mathtt{t_1}) = \alpha_\mathcal{R}(\mathtt{t_2})$. Que la función de abstracción sea compatible con la igualdad de representación es equivalente a afirmar que la igualdad de

la representación está contenida en la igualdad de la abstracción. Así, podemos concluir que la igualdad de representación más pequeña que puede considerarse en una representación, es decir, la que menos elementos identifica, es la propia igualdad del soporte, mientras que la más grande entre las posibles igualdades de representación es la igualdad de la abstracción. Cuando en una representación no se cite explícitamente la igualdad de la representación se entenderá que se está considerando como tal la igualdad del soporte, $=_{\mathcal{S}_\mathcal{R}}$.

El modelo de una representación es lo que un programador tiene en mente y pretende representar.

Definición 3.4.3 *Dado un conjunto A, una* representación de A *es una representación $\mathcal{R}$ tal que $\mathcal{M}_\mathcal{R} = A$.*

A continuación mostramos ejemplos de representaciones.

Ejemplo 3.4.4

1. Una representación del conjunto $\mathbb{Z}$ de los números enteros es $\mathcal{R}_\mathbb{Z}=$ (`integer`, `integer`, = , $\alpha_\mathbb{Z}$, $\mathbb{Z}$) siendo $\alpha_\mathbb{Z}$ la función que a cada entero de Common Lisp `n` le asocia el entero "de las Matemáticas" n.

   En este punto, consideramos adecuada una observación acerca del tipo `integer` y su relación con el conjunto $\mathbb{Z}$. Al igual que ocurre en otros lenguajes de programación, en cualquier Common Lisp existe una cota superior para los datos de tipo `integer`. Debido a la gestión de Common Lisp, esta cota está determinada por el tamaño de la memoria. Así, el mayor dato de tipo `integer` puede hacerse tan grande que es razonable pensar que nadie podría darle un uso matemático aceptable. Números mayores que la cota no habrán sido ni serán usados realmente por ningún matemático (aunque, por supuesto, sí en potencia). En este sentido, podemos decir que hay suficientes enteros en `integer`, al menos potencialmente, para todas las necesidades matemáticas. Es en este sentido en el que pueden considerarse "iguales" ambos conjuntos $\mathbb{Z}$ e `integer`. Otro razonamiento que avala esta última conclusión es pensar en un Common Lisp teórico cuyo único aspecto "teórico" es ése, poder trabajar con enteros tan grandes como se desea. En la práctica, cualquier implementación de Common Lisp en cualquier máquina razonable es una aproximación suficiente a un tal Common Lisp teórico, en el que los resultados teóricos que siguen pueden ser demostrados con todo rigor matemático. Así, podemos considerar la abstracción como función biyectiva.

2. Para el conjunto $\mathbb{N}$ de los números naturales una representación, viéndolo como subconjunto de $\mathbb{Z}$, se obtiene tomando $\mathcal{R}_\mathbb{N}=$ (`integer`, `(satisfies naturalp)`, =, $\alpha_\mathbb{N}$, $\mathbb{N}$) siendo `naturalp` el predicado definido en la página 134 y siendo $\alpha_\mathbb{N}$ la restricción de la función $\alpha_\mathbb{Z}$ del ejemplo anterior al subtipo de `integer` nominado por `(satisfies naturalp)`. Observar que la abstracción de la representación anterior es biyectiva (en el sentido comentado en el punto anterior, relacionado con la sobreyectividad). Otra representación del mismo conjunto $\mathbb{N}$ con el mismo dominio `integer` es tomar todo el conjunto `integer` como soporte; como igualdad, la inducida por la primitiva Lisp `abs` que da el valor absoluto de un dato perteneciente al tipo `integer` (es decir, la dada por la relación de equivalencia `a` $=_\mathcal{R}$ `b` si y solo si `(abs a)` = `(abs b)`, para todo `a`, `b` $\in$ `integer`); y, como función de abstracción la que asocia a cada dato de tipo `integer` su valor absoluto (es decir, la composición de `abs` con $\alpha_\mathbb{Z}$, $\alpha_\mathbb{Z} \circ$ `abs`). Así, la función de abstracción de esta representación no es inyectiva puesto que dos datos concretos del soporte representan al mismo dato abstracto.

3. Fijamos $n \in \mathbb{N}$ con $n > 1$. Consideramos el conjunto cociente $\mathbb{Z}/n\mathbb{Z}$ y tomamos $< n >= \{0, \ldots, n-1\}$ como conjunto de representantes canónicos. Podemos dar varias representaciones naturales para este conjunto, todas ellas con dominio `integer`:

   i) Tomamos como soporte de la representación el subconjunto de `integer` determinado por `(mod n)`(véase [116], página 60); y, como función de abstracción, la restricción de $\alpha_{\mathbb{Z}}$ a `(mod n)`. Esta función de abstracción es biyectiva, así todos los datos del modelo están representados y además por un único dato del soporte.

   ii) Tomamos `integer` como soporte y como función de abstracción la definida por $\alpha_{\mathbb{Z}}$(`d`) módulo n, para cada `d` $\in$ `integer`. Así, la abstracción no es inyectiva aunque sí sobreyectiva, por lo que todos los datos del modelo están representados, pudiendo estarlo por más de un dato del soporte de la representación.

   iii) Tomando `integer` como soporte y como igualdad de representación la definida por la siguiente relación de equivalencia: `a` $=_{\mathcal{R}}$ `b` si `(mod a n) = (mod b n)`, para todo `a`, `b` $\in$ `integer`; la función de abstracción viene dada, para cada `d` $\in$ `integer`, por `d` *mod* n. Al igual que la función de abstracción de la representación anterior, ésta no es inyectiva y sí sobreyectiva.

4. Una representación del conjunto de valores booleanos $Bool = \{true,\ false\}$ es $\mathcal{R}_{Bool} = (\mathcal{U}, \mathcal{U}, \texttt{eq}, \alpha_{Bool}, Bool)$ siendo α_{Bool} la función que al objeto Lisp `nil` le asocia $false$ y asocia $true$ a cualquier otro objeto Lisp. Esta representación coincide con la interpretación booleana que hacen los intérpretes Lisp de los objetos de $\mathcal{U}$.

5. Nos situamos en un universo algebraico que contenga a $\mathbb{N}$ y vamos a dar distintas representaciones para el conjunto de los conjuntos de naturales $\wp(\mathbb{N})$, el conjunto de las partes de $\mathbb{N}$. En todas ellas subyace la primera representación dada para $\mathbb{N}$ en el punto 2. de este mismo ejemplo. Una primera aproximación, muy al estilo Lisp, consistiría en utilizar listas para representar conjuntos de naturales. Sin embargo, hemos preferido comenzar por una representación que sería más común en lenguajes como C o Pascal para acentuar, por una parte, que las propiedades que pretendemos ilustrar no dependen de Common Lisp y, por otra, que Lisp es suficientemente versátil como para adaptarse a un diseño previo (tipo C, Pascal, etc.). En realidad, se va a tener una familia de representaciones parametrizadas por un natural `DIM_MAX` que es el número de componentes que se reservan para el vector que, en todos los ejemplos, va a aparecer en cada elemento del dominio de la representación. Suponemos pues definido

```
(DEFPARAMETER DIM_MAX)
```

o, si preferimos fijar ideas,

```
(DEFPARAMETER DIM_MAX 30)
```

En todo este primer grupo de representaciones que vamos a presentar del conjunto de las partes de $\mathbb{N}$, la base van a ser los datos (ejemplares) de

```
(defstruct CN elementos longitud)
```

tales que `elementos` va a ser un vector de `DIM_MAX` componentes y `longitud` un natural entre `0` y `DIM_MAX`, ambos inclusive. Como es habitual, `longitud` viene a señalar los índices de las componentes del vector `elementos` que son relevantes. Puesto que en Lisp, al igual que en C, los vectores se indexan comenzando en `0`, los índices significativos serán $\{0, \ldots,$ `longitud` $- 1\}$. Si la `longitud` es `0`, se tratará de un dato concreto que representa al conjunto vacío, que en todas las representaciones de esta familia admite como codificación:

```
(make-CN  :elementos  (make-array DIM_MAX)
          :longitud 0)
```

Notar que ésta es la forma elemental, sin usar punteros, más habitual de representar listas simples, homogéneas, en lenguajes como C o Pascal. En la línea expresada anteriormente, vamos a limitarnos a usar arrays simples (de dimensión no ampliable, véase `adjustable-array`, `fill-pointer`, etc., en [116]) y a evitar las posibilidades de la primitiva `vector` que pueden dotar a los vectores Lisp de características muy similares a las listas.

i) Damos una primera representación de este primer grupo, que denotamos por $\mathcal{R}_{\wp(\mathbb{N})}$, cuyo dominio es el dominio `CN` definido anteriormente:

```
(defstruct CN elementos longitud)
```

El soporte de la representación constará de aquellas instancias de tipo `CN` que contengan en el campo `longitud` un dato de tipo `integer` mayor o igual a `0` (el valor `0` corresponderá a codificaciones del conjunto vacío) y cuyo campo `elementos` sea un vector con dimensión mayor o igual que `longitud` que contenga datos de tipo `(satisfies naturalp)` en sus componentes hasta la de índice `longitud` $- 1$. La función de abstracción $\alpha_{\wp(\mathbb{N})}$: `CN` $\rightarrow$ $\wp(\mathbb{N})$ lleva cada dato del soporte al conjunto formado por los naturales almacenados en sus componentes significativas (de la `0` a la `longitud` $- 1$). Observar que, como es habitual, estamos identificando el valor de cada dato de tipo `(satisfies naturalp)` con el natural al que representa según la representación natural de $\mathbb{N}$; es decir, en realidad, la función de abstracción es

$$\alpha_{\wp(\mathbb{N})}(\texttt{c}) = \left\{\alpha_{\mathbb{N}}(\texttt{(aref (CN-elementos c) i)}) \mid \texttt{0} \leq \texttt{i} < \texttt{(CN-longitud c)}\right\}$$

siendo $\alpha_{\mathbb{N}}$ la función de abstracción de la representación de $\mathbb{N}$ definida en el punto 2. de este mismo ejemplo. (Ver en [116] las primitivas de acceso a componentes de vectores y a campos de registros.)

La igualdad específica de Common Lisp para el tipo `CN` (ver más detalles en la página 109 de [116]), viene dada por `equalp`. Sin embargo, definimos como igualdad de representación la siguiente: `x`, `y` $\in \mathcal{S}_{\wp(\mathbb{N})}$, `x` $=_{\mathcal{R}_{\wp(\mathbb{N})}}$ `y` si y solo si `(CN-longitud x) = (CN-longitud y)` y `(aref (CN-elementos x) i) = (aref (CN-elementos y) i)` para todo `0` $\leq$ `i` $<$ `(CN-longitud y)`. Es decir, las componentes significativas de los vectores que forman parte de `x` e `y` deben coincidir, pudiendo ser diferentes el resto. La idea detrás de esta igualdad es que en los datos de tipo `CN` hay información que no interesa y que son las componentes no significativas del vector, y, por tanto, no deben considerarse para identificar objetos. Por ejemplo, hay múltiples instancias de `CN` que representan al conjunto vacío, todas ellas con `longitud 0` con independencia del contenido del vector `elementos`.

Los objetos

```
(make-CN :elementos (make-array DIM_MAX :initial-contents  '(4 3 2 6 1))
         :longitud 3)
```

y

```
(make-CN :elementos (make-array DIM_MAX :initial-contents  '(4 3 2 6 5))
         :longitud 3)
```

están en el soporte $\mathcal{S}_{\wp(\mathbb{N})}$ y son iguales en la representación. Ambos representan al conjunto $\{2,3,4\}$, conjunto también representado por los objetos,

```
(make-CN :elementos (make-array DIM_MAX :initial-contents  '(2 3 4 6 5 8))
         :longitud 3)
```

y

```
(make-CN :elementos (make-array DIM_MAX :initial-contents  '(2 3 4 4 4))
         :longitud 4)
```

Por tanto, los cuatro objetos anteriores son iguales por la abstracción. Así, la abstracción no es inyectiva. Tampoco es sobreyectiva ya que el soporte considerado no permite representar cualquier conjunto de naturales, solo aquellos que tengan cardinal menor o igual que `DIM_MAX` (teniendo en cuenta lo dicho previamente sobre la cota de los enteros en Lisp).

ii) Otra representación $\mathcal{R}^1_{\wp(\mathbb{N})}$ para el mismo conjunto se obtiene restringiendo el soporte $\mathcal{S}_{\wp(\mathbb{N})}$ de la representación $\mathcal{R}_{\wp(\mathbb{N})}$ anterior considerando únicamente aquellos datos del soporte anterior que entre las componentes significativas del vector no contienen elementos repetidos. En este caso se tiene que para que dos objetos `x` e `y` sean iguales por la abstracción, necesariamente deben verificar que `(CN-longitud x) = (CN-longitud y)`.

iii) Otra variante es la representación $\mathcal{R}^2_{\wp(\mathbb{N})}$ obtenida restringiendo el soporte a aquellos datos del soporte anterior $\mathcal{S}_{\wp(\mathbb{N})}$ que poseen las componentes significativas del vector ordenadas, por ejemplo, de forma estrictamente creciente (luego, en particular, no contienen elementos repetidos). En este caso, la igualdad de la representación coincide con la de la abstracción.

Las representaciones anteriores de $\wp(\mathbb{N})$ tienen como restricción que con ellas no hay posibilidad de representar conjuntos infinitos ya que lo que se almacena de cada conjunto es una enumeración de sus elementos. Veamos a continuación otro tipo de representaciones. Damos otra representación de $\wp(\mathbb{N})$ en la que la idea subyacente es codificar un conjunto a través de su función característica. Para ello, es necesario utilizar objetos funcionales como datos concretos de la representación. Definimos una representación que denominamos $\mathcal{R}^f_{\wp(\mathbb{N})}$ que tiene por dominio al tipo de los objetos funcionales. Como soporte de la representación tomamos el conjunto de algoritmos Lisp $\mathcal{A}($`(satisfies naturalp)`$, \mathcal{U})$, siendo $\mathcal{U}$ el universo de objetos Lisp. La función de abstracción $\alpha^f_{\wp(\mathbb{N})}\colon \mathcal{D}^f_{\wp(\mathbb{N})} \to \wp(\mathbb{N})$ viene definida por

$$\alpha^f_{\wp(\mathbb{N})}(\texttt{d}) = \{\alpha_{\mathbb{N}}(\texttt{x}) \in \mathbb{N} \mid \texttt{(not (eq (naturalp x) nil))} \text{ y la evaluación de } \texttt{(funcall d x)} \text{ devuelve un objeto Lisp distinto (por } \texttt{eq}\text{) de } \texttt{nil}\}$$

Como igualdad de la representación se toma la de la abstracción. Así, por ejemplo, el objeto funcional `#'SumaUno` definido en la página 131 es un objeto del soporte anterior y representa al conjunto $\mathbb{N}$. Otro objeto funcional que también representa a $\mathbb{N}$ es el objeto siguiente:

```
#'(lambda (n)  t)
```

Como se observa, este segundo tipo de representaciones permite codificar conjuntos infinitos.

6. Modificando la función de abstracción puede obtenerse de modo sencillo, a partir de la primera de las representaciones del ejemplo anterior, una representación del conjunto de los multiconjuntos de números naturales.

7. Situándonos en el marco de Common Lisp, podemos dar representaciones para $\wp(\texttt{CL})$, el conjunto de conjuntos de objetos Lisp. Con pequeñas variaciones de las representaciones vistas anteriormente para el conjunto de partes de $\mathbb{N}$, obtenemos representaciones de $\wp(\texttt{CL})$.

   i) Una representación, que denotamos $\mathcal{R}_{\wp(\texttt{CL})}$, basada en la representación $\mathcal{R}_{\wp(\mathbb{N})}$, es la que tiene como dominio el conjunto `CCL` de datos de la estructura dada por

   ```
   (defstruct CCL elementos longitud)
   ```

   Como soporte se toman los objetos del dominio anterior que contengan en el campo `longitud` un entero mayor o igual que `0`, en el campo `elementos` un vector y tales que la dimensión del vector sea mayor que el entero almacenado en el campo `longitud`. La función de abstracción $\alpha_{\wp(\texttt{CL})}$ lleva cada dato del soporte anterior al conjunto formado por los objetos Lisp almacenados en las componentes significativas del vector. Finalmente, como igualdad de representación, tomamos aquella que identifica los objetos del soporte con el mismo valor en el campo `longitud` y en las `longitud` primeras componentes del vector `elementos` (considerando la igualdad $=_{\texttt{CL}}$). Al igual que en el caso de los conjuntos de naturales, con pequeñas variantes en la elección del soporte se obtienen distintas representaciones.

   ii) A partir de la representación $\mathcal{R}^f_{\wp(\mathbb{N})}$ dada anteriormente, consideramos una representación del mismo tipo (es decir, una representación cuyos datos son objetos funcionales), que denotamos por $\mathcal{R}^f_{\wp(\texttt{CL})}$, para el conjunto de los conjuntos de objetos Lisp. Como dominio tomamos el tipo de los objetos funcionales; como soporte el conjunto de algoritmos Lisp $\mathcal{A}(\mathcal{U},\mathcal{U})$, siendo $\mathcal{U}$ el universo de objetos Lisp; la función de abstracción $\alpha^f_{\wp(\texttt{CL})} \colon \mathcal{D}^f_{\wp(\texttt{CL})} \to \wp(\texttt{CL})$ es la definida por

   $$\alpha^f_{\wp(\texttt{CL})}(\texttt{d}) = \{\texttt{x} \in \mathcal{U} \mid \text{la evaluación de } \texttt{(funcall d x)} \text{ devuelve un objeto Lisp distinto de } \texttt{nil}\}$$

   y, para completarla, tomamos como igualdad de representación la definida por la función de abstracción.

   Por ejemplo, en la representación anterior el objeto

   ```
   #'(lambda (x)  t)
   ```

   representa al universo $\mathcal{U}$ de objetos Lisp.

8. Sea $\mathcal{L}$ el conjunto de listas formadas por objetos Lisp. A continuación damos una representación $\mathcal{R}_{\mathcal{L}}$ de este conjunto, basada en la representación de una lista mediante la enumeración de sus elementos. El dominio $\mathcal{D}_{\mathcal{L}}$ está formado por el tipo Common Lisp `list`. Antes de determinar el soporte de la representación vamos a hacer algunos comentarios sobre el tipo anterior (para obtener más detalles sobre el tipo `list` se puede consultar [116], páginas 30–31 y capítulo 15). El tipo `list` se define a partir del tipo `cons`. Un objeto de tipo `cons`, consta de dos componentes llamadas *car* y *cdr*. Un objeto pertenece al tipo `list` si es un objeto especial llamado lista vacía o es un cons que en su componente *cdr* es una lista. Así, una lista es una cadena de cons enlazados por la componente *cdr*, almacenándose en las componentes *car* los objetos Lisp que son elementos de la lista. El tipo de datos `list` es la unión de los tipos `cons` y `null` (tipo de datos que contiene como único elemento a `nil`, que representa a la lista vacía y verifica `(eq nil ())`). El tipo `list` engloba a dos clases distintas de objetos. Por una parte, existen objetos de tipo `list` en los que la cadena enlazada de conses que forma el objeto, contiene un cons cuya componente *cdr* apunta a `nil`, y que se denominan *listas "true"*. Por otra, existen objetos en los que todos los cons que forman parte de la lista apuntan a un objeto de cualquier tipo distinto de `nil`, en este caso el objeto puede ser una lista circular (infinita) o lo que se denomina *lista "dotted"*. Tomamos como soporte de la representación $\mathcal{S}_{\mathcal{L}}$ aquellos objetos de tipo `list` que son listas "true". La función de abstracción lleva un objeto del soporte anterior a la lista formada por los objetos Lisp almacenados en las componentes *car* de los conses que forman la lista, además en el mismo orden. Consideramos la igualdad de la abstracción como igualdad de la representación.

9. Siguiendo en la línea de los ejemplos anteriores y pensando ahora en las "listas matemáticas", damos una representación que denotamos $\mathcal{R}^f_{LM}$ utilizando objetos funcionales. Así, tomamos como dominio $\mathcal{D}^f_{LM}$ el tipo de los objetos funcionales; como soporte $\mathcal{S}^f_{LM}$ el conjunto $\mathcal{A}(\texttt{(satisfies naturalp)}, \mathcal{U})$, siendo `naturalp` el predicado definido en la página 134; como función de abstracción la que a cada objeto funcional `d` de $\mathcal{S}^f_{LM}$ le asocia la lista matemática cuyo elemento i-ésimo es `(funcall d i)` (como es natural indexando los elementos de las listas partiendo del índice 0); e igualdad de representación la de la abstracción.

10. Una representación de las aplicaciones de $\mathcal{U}$ en $\mathcal{U}$ que denotamos por $\mathcal{R}^f_{\mathcal{U}^{\mathcal{U}}}$ es la que tiene a $\mathcal{U}$ como dominio; al conjunto de algoritmos Lisp de $\mathcal{U}$ en $\mathcal{U}$, $\mathcal{A}(\mathcal{U},\mathcal{U})$, como soporte; función de abstracción la definida, para cada $\texttt{x} \in \mathcal{A}(\mathcal{U},\mathcal{U})$ por $\alpha^f_{\mathcal{U}^{\mathcal{U}}}(\texttt{x})\colon \mathcal{U} \to \mathcal{U}$ siendo ésta última la función definida por $\alpha^f_{\mathcal{U}^{\mathcal{U}}}(\texttt{x})(\texttt{y}) := \texttt{(funcall x y)}$ (es decir, para cada $\texttt{x} \in \mathcal{A}(\mathcal{U},\mathcal{U})$ es la función definida por el programa $(\texttt{x},\mathcal{U},\mathcal{U},\mathcal{U})$); finalmente, como igualdad de representación tomamos la específica de Common Lisp que en este caso no puede ser otra que `eq`.

11. Si pensamos en representar las aplicaciones de $\mathbb{N}$ en $\mathbb{N}$, una opción es definir la representación $\mathcal{R}_{\mathbb{N}^{\mathbb{N}}}$ cuyo dominio es el tipo `vector`, el soporte está formado por aquellos elementos del dominio anterior cuyas componentes satisfacen todas el predicado `naturalp`, y la función de abstracción viene dada, para cada $\texttt{v} \in \mathcal{S}_{\mathbb{N}^{\mathbb{N}}}$, por la función $\alpha_{\mathbb{N}^{\mathbb{N}}}(\texttt{v})\colon \mathbb{N} \to \mathbb{N}$ definida del siguiente modo

$$\alpha_{\mathbb{N}^{\mathbb{N}}}(\texttt{v})(\texttt{n}) = \begin{cases} \texttt{(aref v n)} & \text{si } n < \texttt{(car (array-dimensions v))} \\ \texttt{0} & \text{en otro caso} \end{cases}$$

donde `(car (array-dimensions v))` es la dimensión del vector `v`. Observar que estamos identificando cada objeto del tipo `(satisfies naturalp)` con el natural al que representa en

la representación $\mathcal{R}_{\mathbb{N}}$, lo que no supone ningún problema ya que la representación es total y la función de abstracción de la representación es considerada biyectiva.

Otra representación para el mismo conjunto, representación que usa objetos funcionales y que denotamos por $\mathcal{R}^f_{\mathbb{N}^{\mathbb{N}}}$, es la que tiene por dominio al tipo de los objetos funcionales, por soporte al conjunto de algoritmos $\mathcal{A}$(`(satisfies naturalp)`, `(satisfies naturalp)`), y por función de abstracción $\alpha^f_{\mathbb{N}^{\mathbb{N}}}$(`x`)(`n`) = `(funcall x n)`, para cada `x` $\in \mathcal{S}^f_{\mathbb{N}^{\mathbb{N}}}$ y cada `n` $\in$ `(satisfies naturalp)`.

□

Para representar modelos cuyos elementos son conjuntos (con o sin estructura añadida) hemos utilizado dos tipos de representaciones. Por un lado, representaciones cuyo dominio está formado por estructuras de almacenamiento (como listas, vectores, registros, etc.) en las que se almacenan las representaciones de los elementos, y que denominaremos *representaciones por extensión*, por remarcar el paralelismo con la definición por extensión de un conjunto. El otro tipo de representaciones están basadas en la noción de *codificación funcional*, noción introducida por Sergeraert en [113] y son las que denominamos *representaciones funcionales*. Las representaciones funcionales son aquellas en las que el dominio está compuesto por objetos funcionales y que reposan en la idea matemática de definición de un conjunto por comprensión. En [65] ya distinguimos entre ambos tipos de representaciones, aunque de un modo más informal.

En los ejemplos anteriores puede observarse que, en el caso de que el conjunto que se pretende representar sea uno para el que existe un tipo de datos predefinido pensado específicamente para él, hay una representación muy natural. Por ejemplo, la representación dada en el ejemplo 3.4.4 para el conjunto $\mathbb{Z}$, es una representación en la que dominio y soporte coinciden, y la función de abstracción es simplemente un cambio de marco de referencia (paso de un lenguaje de programación concreto a las Matemáticas). A las representaciones que dan pasos directos de Common Lisp a las Matemáticas sin ninguna otra transformación añadida las denominaremos *representaciones naturales*.

Un tipo especial de representaciones son las que denominamos *representaciones literales* que son las definidas por la identidad, como función total, sobre un tipo concreto, es decir, tienen la forma $(\mathtt{T}, \mathtt{T}, =_{\mathtt{T}}, id, \mathtt{T})$ con `T` tipo concreto. Denotaremos por $\mathcal{R}^l_{\mathtt{T}}$ a la representación literal del tipo concreto `T`. Notar que no se exige que la igualdad de la representación coincida con la predefinida en Common Lisp para el tipo `T`, sino que debe coincidir con la del propio tipo.

El dominio de una representación establece un patrón para los datos concretos que representan a los del modelo, fija el aspecto de los datos concretos. Como puede observarse en los ejemplos anteriores, un mismo soporte puede utilizarse para representar distintos modelos. Recíprocamente, para un mismo modelo, aún con el mismo soporte, pueden darse distintas representaciones variando la igualdad de la representación y la función de abstracción. La función de abstracción relaciona datos concretos del soporte con datos abstractos del modelo, es decir, se encarga de *decodificar* la información que contiene el soporte. Así, puede pensarse como una función de *decodificación* desde lo concreto (soporte) a lo abstracto (modelo). Relacionadas con propiedades de la función de abstracción podemos considerar propiedades deseables para las representaciones, propiedades que definimos a continuación.

Definición 3.4.5

- *Una representación se dice* plena *si la función de abstracción es sobreyectiva.*
- *Una representación se dice* fiel *si la función de abstracción es inyectiva.*

- *Una representación se dice* total *si la función de abstracción es total, es decir, dominio y soporte de la representación coinciden.*

Si una representación es fiel, cada elemento del modelo está representado, como mucho, por uno del soporte, pudiendo no estarlo por ninguno.

Ejemplo 3.4.6 Consideramos las representaciones introducidas en el ejemplo 3.4.4 y analizamos sus propiedades.

1. La representación natural de $\mathbb{Z}$ es fiel, total y plena pues, como ya hemos insistido, identificamos de modo natural el tipo `integer` con el conjunto $\mathbb{Z}$.

2. La primera representación dada para $\mathbb{N}$ es fiel, plena y no es total; mientras que la segunda no es fiel aunque sí plena y total.

3. Sobre las representaciones dadas para $\mathbb{Z}/n\mathbb{Z}$ con $n > 1$, la definida en *i*) es fiel, plena y no es total; las dadas en *ii*) y *iii*) son plenas y totales aunque no fieles.

4. La representación dada para el conjunto de valores booleanos *Bool* es, evidentemente, plena y total y no fiel.

5. Cualquier representación por extensión de los conjuntos de naturales es claro que no puede ser plena ya que el límite en la dimensión de la estructura de almacenamiento no permite representar conjuntos con cualquier cardinal. De hecho, en todas ellas la función de abstracción no cubre ni siquiera las partes finitas de $\mathbb{N}$. Además, solo la tercera de ellas es fiel. La representación funcional dada en el mismo apartado no es ni total, ni fiel ni plena (partes de $\mathbb{N}$ no es numerable).

6. La representación para los multiconjuntos de naturales no es total, ni fiel, ni plena.

7. La representación por extensión dada para los conjuntos de objetos Lisp así como la funcional, no son ni totales, ni fieles, ni plenas.

8. La representación $\mathcal{R}_{\mathcal{L}}$ no es total, sí fiel y plena (siempre suponiendo un tamaño de memoria que no restrinja la longitud de las listas).

9. La representación $\mathcal{R}^f_{LM}$ tampoco es ni total, ni fiel, ni plena.

10. La representación de las aplicaciones de $\mathcal{U}$ en $\mathcal{U}$ no es total, ni fiel, ni plena.

11. La representación por extensión de las aplicaciones entre naturales $\mathcal{R}_{\mathbb{N}^{\mathbb{N}}}$ es fiel pero no es total ni plena. La funcional no es ni fiel, ni plena, ni total. Observar que si tomásemos como igualdad de representación la de la abstracción, es decir, si considerásemos el cociente del soporte por la equivalencia inducida por la abstracción, la representación funcional sería fiel.

□

Tras dar la noción de representación el siguiente paso es definir los morfismos entre representaciones.

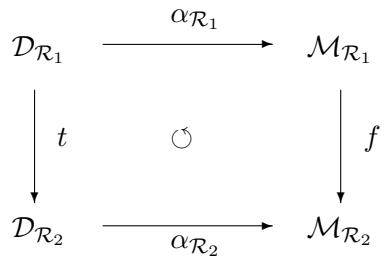
Definición 3.4.7 *Sean* $\mathcal{R}_1 = (\mathcal{D}_{\mathcal{R}_1}, \mathcal{S}_{\mathcal{R}_1}, =_{\mathcal{R}_1}, \alpha_{\mathcal{R}_1}, \mathcal{M}_{\mathcal{R}_1})$ *y* $\mathcal{R}_2 = (\mathcal{D}_{\mathcal{R}_2}, \mathcal{S}_{\mathcal{R}_2}, =_{\mathcal{R}_2}, \alpha_{\mathcal{R}_2}, \mathcal{M}_{\mathcal{R}_2})$ *representaciones. Una* transformación (o *r–transformación*) entre $\mathcal{R}_1$ y $\mathcal{R}_2$ *es un par* (t, f) *donde* $t\colon \mathcal{D}_{\mathcal{R}_1} \to \mathcal{D}_{\mathcal{R}_2}$ *es una función parcial con* $Def(t) = \mathcal{S}_{\mathcal{R}_1}$, $f\colon \mathcal{M}_{\mathcal{R}_1} \to \mathcal{M}_{\mathcal{R}_2}$ *es una función total, y, se verifica:*

i) $t(\mathtt{d}) \in \mathcal{S}_{\mathcal{R}_2}$, *para todo* $\mathtt{d} \in \mathcal{S}_{\mathcal{R}_1}$;

ii) $f(\alpha_{\mathcal{R}_1}(\mathtt{d})) = \alpha_{\mathcal{R}_2}(t(\mathtt{d}))$, *para todo* $\mathtt{d} \in \mathcal{S}_{\mathcal{R}_1}$; *y,*

iii) deben preservarse las igualdades de las representaciones, es decir, si $\mathtt{d}_1 =_{\mathcal{R}_1} \mathtt{d}_2$, *entonces* $t(\mathtt{d}_1) =_{\mathcal{R}_2} t(\mathtt{d}_2)$, *para todo* $\mathtt{d}_1, \mathtt{d}_2 \in \mathcal{S}_{\mathcal{R}_1}$.

En particular, una r–transformación hace conmutativo el siguiente diagrama:



La composición de r–transformaciones se define de manera natural. A partir de las identidades sobre los correspondientes dominios y modelos se define el par (id, id) que es una r–transformación. Así, representaciones y r–transformaciones constituyen una categoría.

A continuación extendemos las nociones de representación y r–transformación al marco de los TADs.

Definición 3.4.8

- *Sea* $A = \left\langle (A_g)_{g\in G}, \{\sigma_A\colon A_\omega \to A_v, Def(\sigma_A)\}_{(\sigma\colon \omega\to v)\in\Sigma}\right\rangle$ *una* Σ*-álgebra. Una* representación de la Σ-álgebra A *es una familia* $\underline{\mathcal{R}}_A = (\mathcal{R}_g)_{g\in G}$ *de representaciones tales que, para cada* $g \in G$, $\mathcal{R}_g$ *es una representación del conjunto soporte* A_g.

- *Sean* A *y* B Σ*-álgebras y sean* $\underline{\mathcal{R}}_A$ *y* $\underline{\mathcal{R}}_B$ *representaciones de* A *y* B *respectivamente. Una* r–transformación de $\underline{\mathcal{R}}_A$ en $\underline{\mathcal{R}}_B$ *es una familia de r–transformaciones* $(t, f) = (t_g, f_g)_{g\in G}$ *siendo, para cada* $g \in G$, *el par* (t_g, f_g) *una r–transformación de* $\mathcal{R}_{A_g}$ *en* $\mathcal{R}_{B_g}$.

Definición 3.4.9 *Una* representación de un TAD $\mathcal{T} = \langle \Sigma, \mathcal{C}(\Sigma)\rangle$ *es una representación de cualquier* Σ*-álgebra que sea objeto de* $\mathcal{C}(\Sigma)$.

Las representaciones de un TAD $\mathcal{T}$ junto con las r–transformaciones entre ellas forman una categoría que denominamos *Categoría de representaciones del TAD* $\mathcal{T}$.

Dada una representación $\underline{\mathcal{R}} = (\mathcal{R}_g)_{g\in G}$ de un TAD $\mathcal{T} = \langle \Sigma, \mathcal{C}(\Sigma)\rangle$, puede extraerse una familia de tipos concretos, indexada por el conjunto de géneros de la signatura Σ, $\underline{\mathtt{T}} = (\mathtt{T}_g)_{g\in G}$, siendo, para cada $g \in G$, $\mathtt{T}_g$ el dominio de la representación $\mathcal{R}_g$. Nos referiremos a $\underline{\mathtt{T}}$ como *G-tipo de la representación* $\underline{\mathcal{R}}$.

Ejemplo 3.4.10 Consideramos la siguiente signatura formada por tres géneros que denotamos *cn*, *nat* y *bool* y por las siguientes operaciones

$$\begin{array}{ccccccc} \mathit{pertenece?} & : & \mathit{nat} & \mathit{cn} & \rightarrow & \mathit{bool} \\ \mathit{union} & : & \mathit{cn} & \mathit{cn} & \rightarrow & \mathit{cn} \\ \mathit{inters} & : & \mathit{cn} & \mathit{cn} & \rightarrow & \mathit{cn} \\ \mathit{vacio?} & : & \mathit{cn} & & \rightarrow & \mathit{bool} \\ \mathit{cardinal} & : & \mathit{cn} & & \rightarrow & \mathit{nat} \\ \mathit{finito?} & : & \mathit{cn} & & \rightarrow & \mathit{bool} \end{array}$$

con la idea de definir TADs cuyas álgebras representen a conjuntos de naturales.

Definimos el tipo abstracto $\mathcal{T}_{\wp(\mathbb{N})}$ sobre la signatura anterior y categoría de álgebras la formada por el álgebra (y sus isomorfas) que tiene como soportes para *cn*, *nat* y *bool* a los conjuntos partes de $\mathbb{N}$, $\mathbb{N}$ y $Bool = \{true, false\}$, respectivamente, y como operaciones las correspondientes operaciones sobre conjuntos: pertenencia, unión, intersección, etc.. Según la definición anterior, una representación de $\mathcal{T}_{\wp(\mathbb{N})}$, en particular del álgebra anterior, será una terna de representaciones $(\mathcal{R}_{\wp(\mathbb{N})}, \mathcal{R}_{\mathbb{N}}, \mathcal{R}_{Bool})$, una de cada uno de los soportes. En particular, podríamos tomar la representación funcional $\mathcal{R}^f_{\wp(\mathbb{N})}$ de $\wp(\mathbb{N})$ vista en el ejemplo 3.4.4. Para el conjunto de los números naturales podemos usar la natural, $\mathcal{R}_{\mathbb{N}}$= (`integer`, `(satisfies naturalp)`, `=`, $\alpha_{\mathbb{N}}$, $\mathbb{N}$), donde $\alpha_{\mathbb{N}}$ es la identificación de un natural Lisp con el correspondiente elemento de $\mathbb{N}$, representación ya introducida en el ejemplo 3.4.4. Como representación para *Bool* tomamos la natural, introducida en el mismo ejemplo: $\mathcal{R}_{Bool} = (\mathcal{U}, \mathcal{U}, \texttt{eq}, \alpha_{Bool}, Bool)$ donde α_{Bool} lleva al dato concreto `nil` (y los que son iguales a él por `eq`, `()`) al abstracto $false$ y al resto los lleva a $true$.

Basándonos en cómo se ha definido el TAD anterior, podríamos definir otros TADs para representar conjuntos cuyos elementos no sean los naturales. Para ello, será necesario añadir otro género a la signatura anterior, género que aparecerá en la aridad de la operación *pertenece*? en lugar del género *nat*, género éste último que se mantendría puesto que aparece como género resultado de la operación *cardinal*. Esto llevaría consigo algunas modificaciones que, básicamente, serían añadir un conjunto soporte para el nuevo género y, por tanto una representación para éste, y sustituir la representación para el soporte del género *cn* por una del soporte adecuado. Por ejemplo, si pensamos en representar conjuntos de objetos Lisp, denotamos por $\mathcal{T}_{\wp(\texttt{CL})}$ al TAD cuya signatura se define a partir de la anterior sustituyendo el género *nat* de la operación *pertenece*? por el género *u* y cuya categoría de álgebras está constituida por el álgebra (y sus isomorfas) con conjuntos soporte para *u*, *cn*, *nat* y *bool*, los conjuntos $\mathcal{U}$ (universo de objetos Lisp), partes de $\mathcal{U}$, $\mathbb{N}$ y Bool, respectivamente. La representación literal de $\mathcal{U}$, junto con la representación funcional dada en el ejemplo 3.4.4 para el conjunto de las partes de $\mathcal{U}$ y las representaciones de $\mathbb{N}$ y *Bool* usadas en la representación del TAD $\mathcal{T}_{\wp(\mathbb{N})}$ dada anteriormente, constituyen una representación del TAD $\mathcal{T}_{\wp(\texttt{CL})}$.

□

3.5 Implementaciones de TADs

Dado un TAD $\mathcal{T} = \langle \Sigma, \mathcal{C}(\Sigma) \rangle$ el objetivo último es disponer de una adecuada codificación de un álgebra de $\mathcal{C}(\Sigma)$. Hasta el momento solo hemos considerado la noción de representación, por tanto, solo nos hemos ocupado de la codificación de los datos, es decir, solo hemos tenido en cuenta los soportes de las Σ-álgebras. Sin embargo, no hemos tenido en cuenta las operaciones, que son las que dotan a las Σ-álgebras de estructura y, por tanto, de capacidad expresiva. Así, el concepto de representación resulta demasiado pobre como paso a la máquina de la estructura matemática de una Σ-álgebra. Se va a enriquecer el concepto de representación considerando las operaciones de las Σ-álgebras, lo que se conseguirá ligando las nociones de representación y de programa.

Debemos obtener programas que trabajando con datos concretos simulen el comportamiento de funciones que trabajan sobre los datos abstractos representados. Concretamente nos preocupamos de simular en la máquina el comportamiento de los operadores que forman parte de un TAD. Es así como aparece la noción de *implementación de un TAD*. En [64] introdujimos una noción de implementación y abordamos un estudio básico sobre ella. En [67] dimos una noción de implementación que presenta algunas variantes respecto a la anterior y que es la que aquí emplearemos.

Hasta ahora, hemos tratado, por un lado con programas y, por otro, con representaciones. Comenzamos introduciendo algunos conceptos que establecen relaciones entre ambos.

Definición 3.5.1 *Un programa* $\mathtt{p} = (\mathtt{c}^{\mathtt{p}}, \mathtt{S}^{\mathtt{p}}, \mathtt{T}_1^{\mathtt{p}}, \ldots, \mathtt{T}_n^{\mathtt{p}}, \mathtt{T}^{\mathtt{p}})$ *se dice* compatible con $n+1$ representaciones $\underline{\mathcal{R}} = (\mathcal{R}_1, \ldots, \mathcal{R}_n, \mathcal{R})$ *si se cumplen las siguientes condiciones:*

i) los dominios de las representaciones, $\mathcal{D}_{\mathcal{R}_1}, \ldots, \mathcal{D}_{\mathcal{R}_n}, \mathcal{D}_{\mathcal{R}}$*, son, respectivamente,* $\mathtt{T}_1^{\mathtt{p}}, \ldots, \mathtt{T}_n^{\mathtt{p}}, \mathtt{T}^{\mathtt{p}}$ *(lo que implícitamente incluye coincidencia en las igualdades de los dominios de las representaciones y los tipos de entrada y salida del programa);*

ii) $\mathtt{S}^{\mathtt{p}}$ *es un subconjunto de* $\mathcal{S}_{\mathcal{R}_1} \times \cdots \times \mathcal{S}_{\mathcal{R}_n}$ *(producto de los soportes de las representaciones);*

iii) la imagen de la función definida por el programa (véase definición 3.3.8) está contenida en el soporte de la representación (es decir, $F(\mathtt{p})(\mathtt{S}^{\mathtt{p}}) \subseteq \mathcal{S}_{\mathcal{R}}$*); y, finalmente,*

iv) el programa $\mathtt{p}$ *es compatible con las igualdades de las representaciones, es decir, para cada* $(\mathtt{d}_1, \ldots, \mathtt{d}_n)$*,* $(\mathtt{d}'_1, \ldots, \mathtt{d}'_n) \in \mathtt{S}^{\mathtt{p}}$*, si* $\mathtt{d}_i =_{\mathcal{R}_i} \mathtt{d}'_i$ *para todo* $i = 1, \ldots, n$*, entonces* $F(\mathtt{p})(\mathtt{d}_1, \ldots, \mathtt{d}_n) =_{\mathcal{R}} F(\mathtt{p})(\mathtt{d}'_1, \ldots, \mathtt{d}'_n)$*.*

Obsérvese que cualquier programa $\mathtt{p} = (\mathtt{c}^{\mathtt{p}}, \mathtt{S}^{\mathtt{p}}, \mathtt{T}_1^{\mathtt{p}}, \ldots, \mathtt{T}_n^{\mathtt{p}}, \mathtt{T}^{\mathtt{p}})$ es compatible con las representaciones literales de sus tipos de entrada y salida $\mathcal{R}^l_{\mathtt{T}_1^{\mathtt{p}}}, \ldots, \mathcal{R}^l_{\mathtt{T}_n^{\mathtt{p}}}, \mathcal{R}^l_{\mathtt{T}^{\mathtt{p}}}$. Además, al exigir compatibilidad del programa con la igualdad de la representación, podemos asegurar que $F(\mathtt{p})$ está bien definida sobre el cociente inducido en $\mathtt{S}^{\mathtt{p}}$ por dicha igualdad. En la misma línea, la siguiente definición permitirá el ascenso de funciones hasta el nivel de los modelos.

Definición 3.5.2 *Un programa* $\mathtt{p} = (\mathtt{c}^{\mathtt{p}}, \mathtt{S}^{\mathtt{p}}, \mathtt{T}_1^{\mathtt{p}}, \ldots, \mathtt{T}_n^{\mathtt{p}}, \mathtt{T}^{\mathtt{p}})$ *se dice* coherente con $n+1$ representaciones $\underline{\mathcal{R}} = (\mathcal{R}_1, \ldots, \mathcal{R}_n, \mathcal{R})$ *si el programa* $\mathtt{p}$ *es compatible con las representaciones y, también lo es con las igualdades de las abstracciones. Esta última condición se traduce en que si* $(\mathtt{d}_1, \ldots, \mathtt{d}_n), (\mathtt{d}'_1, \ldots, \mathtt{d}'_n) \in \mathtt{S}^{\mathtt{p}}$ *son tales que* $\alpha_{\mathcal{R}_i}(\mathtt{d}_i) = \alpha_{\mathcal{R}_i}(\mathtt{d}'_i)$ *para todo* $i = 1, \ldots, n$*, entonces*

$$\alpha_{\mathcal{R}}(F(\mathtt{p})(\mathtt{d}_1, \ldots, \mathtt{d}_n)) = \alpha_{\mathcal{R}}(F(\mathtt{p})(\mathtt{d}'_1, \ldots, \mathtt{d}'_n)).$$

Es evidente que todo programa también es coherente con las representaciones literales de sus tipos de entrada y salida.

A partir de $n+1$ representaciones y de un programa $\mathtt{p} = (\mathtt{c}^{\mathtt{p}}, \mathtt{S}^{\mathtt{p}}, \mathtt{T}_1^{\mathtt{p}}, \ldots, \mathtt{T}_n^{\mathtt{p}}, \mathtt{T}^{\mathtt{p}})$ coherente con ellas, podemos definir una función sobre los modelos de las representaciones tal y como recogemos a continuación.

Definición 3.5.3 *Sea* $\mathtt{p} = (\mathtt{c}^{\mathtt{p}}, \mathtt{S}^{\mathtt{p}}, \mathtt{T}_1^{\mathtt{p}}, \ldots, \mathtt{T}_n^{\mathtt{p}}, \mathtt{T}^{\mathtt{p}})$ *un programa coherente con* $n+1$ *representaciones* $\underline{\mathcal{R}} = (\mathcal{R}_1, \ldots, \mathcal{R}_n, \mathcal{R})$. *La* función definida por $\mathtt{p}$ a través de $\underline{\mathcal{R}}$ *es la función* $F(\mathtt{p})_{\underline{\mathcal{R}}} \colon \mathcal{M}_{\mathcal{R}_1} \times \cdots \times \mathcal{M}_{\mathcal{R}_n} \to \mathcal{M}_{\mathcal{R}}$ *con dominio de definición*

$$Def\big(F(\mathtt{p})_{\underline{\mathcal{R}}}\big) = \big\{(m_1, \ldots, m_n) \in \mathcal{M}_{\mathcal{R}_1} \times \cdots \times \mathcal{M}_{\mathcal{R}_n} \;\mid\; \exists (\mathtt{d}_1, \ldots, \mathtt{d}_n) \in \mathtt{S}^{\mathtt{p}} \text{ con } \alpha_{\mathcal{R}_i}(\mathtt{d}_i) = m_i, \text{ para todo } i = 1, \ldots, n\big\}$$

y con el siguiente comportamiento

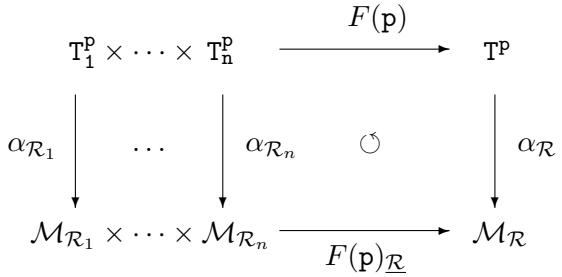
$$F(\mathtt{p})_{\underline{\mathcal{R}}}(\alpha_{\mathcal{R}_1}(\mathtt{d}_1), \ldots, \alpha_{\mathcal{R}_n}(\mathtt{d}_n)) := \alpha_{\mathcal{R}}(F(\mathtt{p})(\mathtt{d}_1, \ldots, \mathtt{d}_n)).$$

Otra forma de expresar $Def\big(F(\mathtt{p})_{\underline{\mathcal{R}}}\big)$ es $Im(\alpha_{\mathcal{R}_1} \times \cdots \times \alpha_{\mathcal{R}_n})(\mathtt{S}^{\mathtt{p}})$.

Notar que $F(\mathtt{p})_{\underline{\mathcal{R}}}$ está bien definida, es decir, es una función, porque $\mathtt{p}$ es compatible con las igualdades de las abstracciones (ya que si para un elemento $(m_1, \ldots, m_n) \in Def(F(\mathtt{p})_{\underline{\mathcal{R}}})$ existen $(\mathtt{d}_1, \ldots, \mathtt{d}_n), (\mathtt{d}'_1, \ldots, \mathtt{d}'_n) \in \mathtt{S}^{\mathtt{p}}$ tales que $\alpha_{\mathcal{R}_i}(\mathtt{d}_i) = \alpha_{\mathcal{R}_i}(\mathtt{d}'_i) = m_i$ para todo $i = 1, \ldots, n$, por ser el programa coherente con las representaciones se verifica que $\alpha_{\mathcal{R}}(F(\mathtt{p})(\mathtt{d}_1, \ldots, \mathtt{d}_n)) = \alpha_{\mathcal{R}}(F(\mathtt{p})(\mathtt{d}'_1, \ldots, \mathtt{d}'_n)))$.

Un caso particular de la definición anterior es el caso en el que las representaciones consideradas son las representaciones literales $\underline{\mathcal{R}}^l$. En este caso, la función definida por el programa a través de $\underline{\mathcal{R}}^l$, $F(\mathtt{p})_{\underline{\mathcal{R}}^l}$, coincide con $F(\mathtt{p})$, función definida por el programa. Así, $F(\mathtt{p})_{\underline{\mathcal{R}}^l} = F(\mathtt{p})$.

La idea subyacente en la definición anterior es obtener de un programa $\mathtt{p}$ coherente con las representaciones $\underline{\mathcal{R}}$, que se encuentra en el nivel concreto del lenguaje de programación, una función en el nivel conceptual, la función $F(\mathtt{p})_{\underline{\mathcal{R}}}$, cuyo comportamiento abstraiga el del programa. Teniendo en cuenta que el paso entre los niveles concreto y abstracto se realiza a través de la función de abstracción, es natural que ésta esté involucrada en la definición de la función $F(\mathtt{p})_{\underline{\mathcal{R}}}$. Por definición de la función $F(\mathtt{p})_{\underline{\mathcal{R}}}$ se tiene la conmutatividad del siguiente diagrama:



El dominio de la función $F(\mathtt{p})_{\underline{\mathcal{R}}}$ está formado por aquellos elementos que están representados por, al menos, un dato del dominio de definición del programa $\mathtt{p}$. Fijarse que la idea que

subyace es que $F(\mathtt{p})_{\underline{\mathcal{R}}}$ represente en el nivel abstracto, el de los modelos, el comportamiento de $F(\mathtt{p})$, mejor dicho de $\mathtt{p}$, en el nivel de lo concreto (el universo del computador). Como la función $F(\mathtt{p})_{\underline{\mathcal{R}}}$ debe simular el comportamiento de la función $F(\mathtt{p})$ y el dominio de ésta es el de definición del programa $\mathtt{p}$, para que un dato esté en el dominio de $F(\mathtt{p})_{\underline{\mathcal{R}}}$ no es suficiente con que esté representado por un dato del producto de los soportes (los significativos para las representaciones de los tipos de entrada del programa) sino que debe estarlo por uno del dominio de definición del programa. La relación entre las funciones $F(\mathtt{p})$ y $F(\mathtt{p})_{\underline{\mathcal{R}}}$, expresada en el diagrama anterior, nos va a permitir concretar el significado de "programa que implementa una función".

Definición 3.5.4 *Sea $f\colon A_1 \times \cdots \times A_n \to A$ una función y sean $\underline{\mathcal{R}} = (\mathcal{R}_1, \ldots, \mathcal{R}_n, \mathcal{R})$ $n+1$ representaciones de $A_1, \ldots, A_n, A$, respectivamente. Se dice que f es* calculable respecto a $\underline{\mathcal{R}}$ *si existe un programa $\mathtt{p}$ coherente con las representaciones $\underline{\mathcal{R}}$ y que verifica las siguientes condiciones:*

i) $Def(F(\mathtt{p})_{\underline{\mathcal{R}}}) \subseteq Def(f)$*;*

ii) $F(\mathtt{p})_{\underline{\mathcal{R}}} = f|_{Def(F(\mathtt{p})_{\underline{\mathcal{R}}})}$*; y,*

iii) para cada $(a_1, \ldots, a_n) \in Def(f)$ tal que existe $(\mathtt{d}_1, \ldots, \mathtt{d}_n) \in \mathcal{S}_{\mathcal{R}_1} \times \cdots \times \mathcal{S}_{\mathcal{R}_n}$ con $\alpha_{\mathcal{R}_i}(\mathtt{d}_i) = a_i$ para todo $i = 1, \ldots, n$ verificando que $f(a_1, \ldots, a_n) \in Im(\alpha_{\mathcal{R}})$, se tiene que existe $(\mathtt{d}'_1, \ldots, \mathtt{d}'_n) \in \mathtt{S}^{\mathtt{P}}$ con $\alpha_{\mathcal{R}_i}(\mathtt{d}'_i) = a_i$, para todo $i = 1, \ldots, n$.

En este caso diremos que $\mathtt{p}$ implementa a f a través de las representaciones $\underline{\mathcal{R}}$ *o que* $\mathtt{p}$ *es una* implementación de f a través de $\underline{\mathcal{R}}$.

El comportamiento exigido a un programa $\mathtt{p} = (\mathtt{c}^{\mathtt{P}}, \mathtt{S}^{\mathtt{P}}, \mathtt{T}_1^{\mathtt{P}}, \ldots, \mathtt{T}_n^{\mathtt{P}}, \mathtt{T}^{\mathtt{P}})$ que implementa a una función $f\colon A_1 \times \cdots \times A_n \to A$ a través de unas representaciones $\underline{\mathcal{R}} = (\mathcal{R}_1, \ldots, \mathcal{R}_n, \mathcal{R})$, hace que se verifique que, para todo $(\mathtt{d}_1, \ldots, \mathtt{d}_n) \in \mathtt{S}^{\mathtt{P}}$:

$$\alpha_{\mathcal{R}}(F(\mathtt{p})(\mathtt{d}_1, \ldots, \mathtt{d}_n)) = f(\alpha_{\mathcal{R}_1}(\mathtt{d}_1), \ldots, \alpha_{\mathcal{R}_n}(\mathtt{d}_n))$$

Como se desprende de lo anterior, se trata de que la función definida por el programa a través de las representaciones sea lo más "parecida" posible a la función a la que implementa. La igualdad entre $Def(f)$ y $Def(F(\mathtt{p})_{\underline{\mathcal{R}}})$ es, en general, una exigencia demasiado fuerte e innecesaria: no parece razonable exigir la igualdad de sus dominios cuando estamos manejando representaciones que no necesariamente son plenas. En este sentido, la condición recogida en *ii*) asegura que sobre aquellos datos abstractos que están representados por datos concretos, el comportamiento de la función definida por el programa a través de las representaciones, es el mismo que el de la función que se está implementando. No es suficiente que sobre los datos de la intersección de los dominios de f y $F(\mathtt{p})_{\underline{\mathcal{R}}}$ ambas se comporten de la misma manera, sino que los dominios deben guardar alguna relación, por ello se imponen las condiciones *i*) y *iii*). La primera de ellas asegura que todos los elementos del soporte del programa representan a datos del dominio de definición de la función que se está implementando, luego establece una cota superior del dominio del programa. Por su parte, la condición *iii*) exige que todos los datos del dominio de la función representados y cuya imagen por la función también lo está, deben estar representados por alguno del soporte del programa. Esta condición marca cuáles son los datos que como mínimo deben estar representados para estar implementando la función y, cubre también, el caso de desbordamiento o salidas de rango al incluir la exigencia de que la imagen por la función también esté representada.

Una definición más compacta, aunque menos clara, de función calculable respecto a $n+1$ representaciones $\underline{\mathcal{R}}$ se obtiene tomando *ii*) y la siguiente igualdad entre dominios:

$$Def(F(\mathtt{p})_{\underline{\mathcal{R}}}) = Def(f) \cap Im(\alpha_{\mathcal{R}_1} \times \cdots \times \alpha_{\mathcal{R}_n}) \cap f^{-1}(Im(\alpha_{\mathcal{R}}))$$

Cuando una función f sea calculable respecto a las representaciones literales de su dominio y codominio, diremos simplemente que f es *calculable*. En este caso, si $\mathtt{p}$ es un programa que implementa a f coherente con las representaciones literales, diremos, simplemente, que $\mathtt{p}$ es una *implementación de* f.

Vamos a extender las definiciones anteriores al contexto algebraico.

Definición 3.5.5 *Sea $\Sigma = \langle G, \Omega\rangle$ una signatura. Una* implementación de una Σ-álgebra A *es un par $\mathcal{I}_A = (\underline{\mathcal{R}}_A, (\mathtt{p}_\sigma)_{\sigma\in\Omega})$ donde $\underline{\mathcal{R}}_A$ es una representación de A y la familia de programas $(\mathtt{p}_\sigma)_{\sigma\in\Omega}$ implementa las operaciones de A a través de $\underline{\mathcal{R}}_A$.*

Observar que en la definición anterior de implementación está incluida la condición de que los programas deben ser coherentes con las representaciones. Extendemos la definición anterior al marco de los TADs.

Definición 3.5.6 *Una* implementación de un TAD $\mathcal{T} = \langle \Sigma, \mathcal{C}(\Sigma)\rangle$ *es una implementación de cualquier Σ-álgebra de $\mathcal{C}(\Sigma)$.*

La propiedad recogida a continuación permite obtener implementaciones a partir de otras implementaciones.

Proposición 3.5.7 *Si $\mathcal{I} = (\underline{\mathcal{R}}, (\mathtt{p}_\sigma)_{\sigma\in\Omega})$ es una implementación de un TAD $\mathcal{T}$ y $(\mathtt{q}_\sigma)_{\sigma\in\Omega}$ son programas coherentes con $\underline{\mathcal{R}}$ y tales que $F(\mathtt{p}_\sigma) = F(\mathtt{q}_\sigma)$, para todo $\sigma \in \Omega$, se tiene que el par $(\underline{\mathcal{R}}, (\mathtt{q}_\sigma)_{\sigma\in\Omega})$ también es una implementación de $\mathcal{T}$.*

Observar que lo que se exige es que, para cada operación $\sigma \in \Omega$, las funciones definidas por los programas $\mathtt{p}_\sigma$ y $\mathtt{q}_\sigma$ sean la misma, lo que en particular conlleva que tengan el mismo dominio.

A continuación introducimos la noción de morfismo entre implementaciones. En [64] introdujimos y trabajamos con una variante de ella.

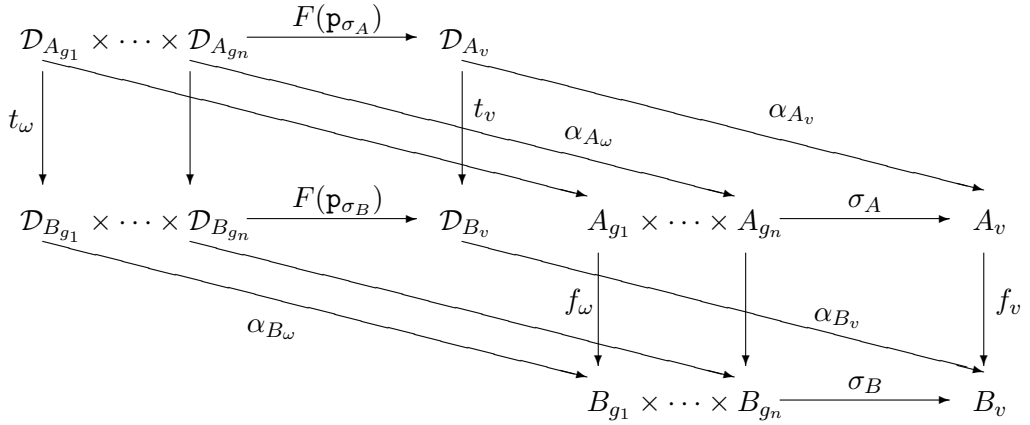
Definición 3.5.8 *Sea $\mathcal{T} = \langle \Sigma, \mathcal{C}(\Sigma)\rangle$ un TAD, sean A y B dos Σ-álgebras de $\mathcal{C}(\Sigma)$ y sean $\mathcal{I}_A = (\underline{\mathcal{R}}_A, (\mathtt{p}_{\sigma_A})_{\sigma\in\Omega})$ e $\mathcal{I}_B = (\underline{\mathcal{R}}_B, (\mathtt{p}_{\sigma_B})_{\sigma\in\Omega})$ implementaciones de A y B respectivamente. Una* i–transformación entre $\mathcal{I}_A$ e $\mathcal{I}_B$ *es una r–transformación $(t, f) = (t_g, f_g)_{g\in G}$ de $\underline{\mathcal{R}}_A$ en $\underline{\mathcal{R}}_B$ con $f\colon A \to B$ un Σ-homomorfismo total verificando, para cada operación $\sigma\colon \omega \to v$ de Ω, las dos condiciones siguientes*

i) $t_\omega(Def(F(\mathtt{p}_{\sigma_A}))) \subseteq Def(F(\mathtt{p}_{\sigma_B}))$, donde, como es natural, t_ω denota a la aplicación $(t_{g_1}, \ldots, t_{g_n})$ siendo $\omega = g_1 \ldots g_n$;

ii) para cada $d_\omega \in Def(F(\mathtt{p}_{\sigma_A}))$ se tiene que

$$t_v(F(\mathtt{p}_{\sigma_A})(\mathtt{d}_\omega)) = F(\mathtt{p}_{\sigma_B})(t_\omega(\mathtt{d}_\omega))$$

Denotaremos por $(t, f)\colon \mathcal{I}_A \to \mathcal{I}_B$ a una i–transformación de $\mathcal{I}_A$ a $\mathcal{I}_B$. Las condiciones de la definición anterior se traducen en que todas las caras del siguiente diagrama conmutan.



Vamos a hacer algunos comentarios sobre la definición anterior. En primer lugar, observar que todas las funciones consideradas, excepto el homomorfismo f, son funciones parciales, lo que debe tenerse en cuenta al exigir un determinado comportamiento a la i–transformación. En particular, la conmutatividad del diagrama (por hipótesis todas las caras del retículo anterior son conmutativas excepto la que se exige en la definición) deberemos exigirla sobre los elementos comunes al dominio de definición de las funciones $t_v \circ F(\mathtt{p}_{\sigma_A})$ y $F(\mathtt{p}_{\sigma_B}) \circ t_\omega$. Notar que como $\mathtt{p}_{\sigma_A}$ es coherente con $\underline{\mathcal{R}}_{A_\omega}$ (siendo, como es natural $\underline{\mathcal{R}}_{A_\omega} = (\mathcal{R}_{A_{g_1}}, \ldots, \mathcal{R}_{A_{g_n}})$, si $\omega = g_1 \ldots g_n$), el dominio de definición de $t_v \circ F(\mathtt{p}_{\sigma_A})$ es $\mathtt{S}^{\mathtt{p}_{\sigma_A}}$, soporte del programa $\mathtt{p}_{\sigma_A}$, (puesto que $Im(F(\mathtt{p}_{\sigma_A})) \subseteq \mathcal{S}_{A_v}$). Sin embargo, ninguna hipótesis garantiza que $t_\omega(\mathtt{S}^{\mathtt{p}_{\sigma_A}}) \subseteq \mathtt{S}^{\mathtt{p}_{\sigma_B}}$ (ya que dado $\mathtt{d}_\omega \in \mathtt{S}^{\mathtt{p}_{\sigma_A}}$, lo único que puede asegurarse es que $\exists \mathtt{d}'_\omega \in \mathcal{S}_{A_\omega}$ con $\alpha_{A_{g_i}}(\mathtt{d}_i) = \alpha_{A_{g_i}}(\mathtt{d}'_i)$ para todo $i = 1, \ldots, n$ verificando que $t_\omega(\mathtt{d}'_\omega) \in \mathtt{S}^{\mathtt{p}_{\sigma_B}}$, pero $t_\omega(\mathtt{d}_\omega)$ puede no estar en $\mathtt{S}^{\mathtt{p}_{\sigma_B}}$, soporte del programa $\mathtt{p}_{\sigma_B}$). Esto explica la razón de imponer la condición i). Observar también que como estamos con implementaciones, las funciones que aparecen en las condiciones son las definidas por los programas, las funciones $F(\mathtt{p}_{\sigma_A})$ y no las definidas a través de las representaciones $F(\mathtt{p}_{\sigma_A})_{\underline{\mathcal{R}}_{A_\omega}}$.

Proposición 3.5.9 *Sea $(t, f)\colon \mathcal{I}_A \to \mathcal{I}_B$ una i–transformación entre dos implementaciones $\mathcal{I}_A = (\underline{\mathcal{R}}_A, (\mathtt{p}_{\sigma_A})_{\sigma\in\Omega})$ e $\mathcal{I}_B = (\underline{\mathcal{R}}_B, (\mathtt{p}_{\sigma_B})_{\sigma\in\Omega})$ de un TAD $\mathcal{T}$. Se tienen las siguientes propiedades:*

i) Si $(a_1, \ldots, a_n) \in Def(F(\mathtt{p}_{\sigma_A})_{\underline{\mathcal{R}}_A})$, entonces $(f_{g_1}(a_1), \ldots, f_{g_n}(a_n)) \in Def(F(\mathtt{p}_{\sigma_B})_{\underline{\mathcal{R}}_B})$.

ii) Si $(\mathtt{d}_1, \ldots, \mathtt{d}_n) \in Def(F(\mathtt{p}_{\sigma_A}))$, entonces se tiene $(\alpha_{B_{g_1}}(t_{g_1}(\mathtt{d}_1)), \ldots, \alpha_{B_{g_n}}(t_{g_n}(\mathtt{d}_n))) \in Def(F(\mathtt{p}_{\sigma_B})_{\underline{\mathcal{R}}_B}))$.

Las i–transformaciones pueden componerse, y existe la i–transformación identidad. Así, dado un TAD $\mathcal{T}$ podemos considerar la *Categoría de implementaciones del TAD $\mathcal{T}$*, categoría que denotamos por $Imp(\mathcal{T})$, con objetos las implementaciones de $\mathcal{T}$ y con morfismos las i– transformaciones entre ellas.

A partir de cada implementación $\mathcal{I} = (\underline{\mathcal{R}}, (\mathtt{p}_\sigma)_{\sigma\in\Omega})$ puede obtenerse una familia de conjuntos $\underline{\mathtt{S}} = (\mathtt{S}^{\mathtt{p}_\sigma})_{\sigma\in\Omega}$ siendo cada $\mathtt{S}^{\mathtt{p}_\sigma}$ el dominio de definición del programa $\mathtt{p}_\sigma$. A la familia $\underline{\mathtt{S}}$ la denominamos Ω-*conjunto de la implementación*. En el caso de que una operación σ sea constante, el dominio de definición del programa $\mathtt{p}_\sigma$ será el tipo `nil` (tipo vacío).

3.6 Decidibilidad de las representaciones

En secciones anteriores hemos estudiado la noción de *representación*. En ella se establece la relación entre lo que hemos denominado datos abstractos y datos concretos, y es una de las bases para definir el concepto de implementación. Hay una propiedad de las representaciones que no hemos mencionado todavía y que tiene gran interés: la decidibilidad. Además, como veremos, esta propiedad implica algunas consecuencias sobre las implementaciones que se pueden definir a partir de una representación. En [65] incluimos un análisis de esta propiedad de las representaciones, análisis menos detallado que el que llevamos a cabo a continuación y en un contexto que varía ligeramente. Para realizar este análisis nos hemos apoyado en [102], siendo los resultados que aquí presentamos traducciones al marco de los TADs de resultados que aparecen en ese trabajo.

Uno de los elementos de una representación es el soporte, definido únicamente como subconjunto del dominio, sin exigirle ninguna otra condición adicional. Esto da pie a considerar la siguiente definición.

Definición 3.6.1 *Se dice que una representación es* decidible *si el soporte es un subconjunto decidible del dominio.*

Relacionado directamente con la noción de soporte está lo que se conoce como invariante de la representación, que es la función característica del soporte como subconjunto del dominio. De hecho, la definición de representación dada por Hoare en [58] utiliza el invariante en lugar del soporte. Así, la decidibilidad de una representación equivale a la calculabilidad de su invariante. El hecho de que el invariante pueda ser o no calculable es lo que nos ha llevado a usar en la definición de representación el soporte, en lugar de utilizar el invariante. De esta forma queda más patente la posibilidad de que el soporte puede ser indecidible.

En Teoría de la Calculabilidad una función se dice calculable si existe un algoritmo que la implementa. En la sección anterior hemos introducido nuestra noción de función calculable pensada para el marco que estamos construyendo. Teniendo en cuenta la noción de programa que hemos dado, es fácil ver que ambas nociones de función calculable coinciden, solo que la nuestra adaptada al marco en el que nos movemos que es Common Lisp. Observar que el invariante es una función que vive en Common Lisp $\texttt{inv}\colon \mathcal{D} \to \mathcal{U}$, por lo que es posible hablar de programa que lo implementa a través de las representaciones literales, lo que nos permite hablar de implementación del invariante sin más.

Por ejemplo, la representación del conjunto de los conjuntos de naturales $\mathcal{R}_{\wp(\mathbb{N})}$ dada en el ejemplo 3.4.4 es decidible. El invariante $\texttt{inv}_{\wp(\mathbb{N})}\colon \mathcal{D}_{\wp(\mathbb{N})} \to \mathcal{U}$ viene dado por $\texttt{inv}_{\wp(\mathbb{N})}(\texttt{n}) := \texttt{t}$ si el campo `longitud` del dato `n` pertenece al tipo `(satisfies naturalp)`, el campo `elementos` es un vector de dimensión mayor o igual que `longitud`, el dato almacenado en el campo `longitud` es un entero mayor o igual que `0`, y en las componentes de este vector (de la `0` a la `longitud-1`) se almacenan datos de tipo `(satisfies naturalp)`; $\texttt{inv}_{\wp(\mathbb{N})}(\texttt{d}) := \texttt{nil}$ en otro caso. Un programa que implementa a la función $\texttt{inv}_{\wp(\mathbb{N})}$ es el programa $\texttt{p}_{\texttt{inv}_{\wp(\mathbb{N})}} = (\texttt{c}^{\texttt{p}_{\texttt{inv}_{\wp(\mathbb{N})}}}, \mathcal{D}_{\wp(\mathbb{N})}, \mathcal{D}_{\wp(\mathbb{N})}, \mathcal{U})$ donde $\texttt{c}^{\texttt{p}_{\texttt{inv}_{\wp(\mathbb{N})}}}$ es el siguiente objeto funcional, objeto que es un algoritmo de $\mathcal{D}_{\wp(\mathbb{N})}$ en $\mathcal{U}$:

```
#'(lambda (x)
     (and (indicep (CN-longitud x))
          (vectorp (CN-elementos x))
          (< (CN-longitud x) (car (array-dimensions (CN-elementos x))))
          (vectorNat (CN-elementos x) (CN-longitud x))))
```

siendo `indicep` el predicado definido a continuación que decide si el valor del campo `longitud` está en el rango de índices válido para el vector almacenado en el campo `elementos`

```
(defun indicep (n)
       (and (integerp n) (>= n 0)))
```

y siendo `vectorNat` el siguiente código, que comprueba si en las componentes significativas de un vector se han almacenado datos de tipo `(satisfies naturalp)`

```
(defun vectorNat(v n)
       (do ((i 0))
           ((OR (= i n) (not (naturalp (aref v i)))) (= i n))
         (setq i (+ i 1))))
```

Análogamente ocurre con la representación por extensión para los conjuntos de objetos Lisp $\mathcal{R}_{\texttt{CCL}}$ dada en 3.4.4, la existencia del siguiente algoritmo de $\mathcal{D}_{\texttt{CCL}}$ en $\mathcal{U}$ muestra que la representación es decidible

```
#'(lambda (x)
      (and (indicep (CCL-num x))
           (vectorp (CCL-elementos x))
           (< (CCL-num x) (car (array-dimensions (CCL-elementos x))))))
```

siendo `indicep` el predicado utilizado en el estudio de la decidibilidad de $\mathcal{R}_{\wp(\mathbb{N})}$ y que calcula si el valor del campo `longitud` está en el rango de índices válido para el vector almacenado en el campo `elementos`.

Las dos representaciones anteriores son representaciones por extensión, sus dominios son estructuras de almacenamiento en cuyas componentes están las codificaciones de los elementos del conjunto. Es natural que ambas sean decidibles ya que para que un dato esté en el soporte solo hay que tener en cuenta las salidas de rango y si los objetos almacenados en las estructuras de almacenamiento pertenecen a tipos adecuados. Sin embargo, es habitual encontrar problemas en los que el cardinal de los conjuntos a representar hace imposible su representación por extensión. Como solución a este problema Segeraert propuso el uso de representaciones funcionales [113], es decir, tomar representaciones en las que los dominios estén formados por objetos funcionales. En el caso de los conjuntos, como ya hemos visto en el ejemplo 3.4.4, se trata de codificar un conjunto por medio de su función característica. Pese a que el uso de representaciones funcionales amplía el número de conjuntos que pueden codificarse, este tipo de representaciones presenta limitaciones en otros aspectos. Por un lado, las representaciones funcionales no poseen una característica deseable, la *decidibilidad*, nunca existirá un programa que decida si un elemento del dominio funcional pertenece o no al soporte. Por otro, solo permiten obtener información local sobre cada objeto representado, lo que provoca que los operadores del tipo abstracto *conjunto* que proporcionan información global no puedan ser implementados. Serán por tanto representaciones *localmente efectivas*, según la terminología de Sergeraert (véase [101]). Vamos a analizar más detalladamente estas dos cuestiones. Lo haremos sobre representaciones funcionales de conjuntos que, aunque solo se trata de un caso particular, es bastante representativo de la forma en la que trabajaremos con los tipos abstractos que corresponden a las diferentes estructuras algebraicas que aparecen en EAT.

3.6.1 Representaciones indecidibles

Comenzamos mostrando que la representación funcional de conjuntos es indecidible. Por simplicidad vamos a demostrarlo para conjuntos de objetos Lisp. Recordamos la representación funcional $\mathcal{R}^f_{\wp(\texttt{CL})}$ para el conjunto de los conjuntos de objetos Lisp dada en el ejemplo 3.4.4: el dominio está formado por el tipo de los objetos funcionales; el soporte por el conjunto de algoritmos Lisp $\mathcal{A}(\mathcal{U},\mathcal{U})$ (interpretando todo lo que es distinto de `nil` como `t`, podemos hablar del conjunto de predicados Lisp); la función de abstracción $\alpha^f_{\wp(\texttt{CL})}\colon \mathcal{D}^f_{\wp(\texttt{CL})} \to \wp(\texttt{CL})$ viene dada por $\alpha^f_{\wp(\texttt{CL})}(\texttt{d}) = \{\texttt{x} \in \mathcal{U} \mid \texttt{(not (eq (funcall d x) nil))}\}$, es decir, la imagen de un objeto Lisp `d` es el conjunto formado por aquellos objetos Lisp `x` tales que la evaluación de `(funcall d x)` devuelve un objeto distinto (por `eq`) de `nil`; como igualdad de representación se considera la definida por la abstracción.

El teorema que sigue es una versión Common Lisp debida a Sergeraert (véanse [100], página 45, y [102], página 16) de la Paradoja de Russell sobre el conjunto de los conjuntos que no pertenecen a sí mismos.

Teorema 3.6.2 *La representación funcional de conjuntos de objetos Lisp $\mathcal{R}^f_{\wp(\texttt{CL})}$ (dada en el ejemplo 3.4.4) es indecidible.*

Demostración:

Para demostrarlo vamos a emplear la misma técnica que en la demostración original de la paradoja de Russell, por lo que utilizamos un objeto, que llamamos `paradoja`, que jugará el mismo papel que el conjunto de los conjuntos que no pertenecen a sí mismos. Notar que cada elemento de $\mathcal{S}^f_{\wp(\texttt{CL})}$ es un predicado Lisp (o lo que es lo mismo un algoritmo de $\mathcal{U}$ en $\mathcal{U}$). Suponer que el invariante es calculable, lo que significa asumir la existencia de un algoritmo `inv?` que decide si un elemento del dominio (`functionp`) es o no un predicado Lisp. Por tanto, se tiene que:

$$\texttt{(funcall inv? objeto)} = \begin{cases} \texttt{t} & \text{si } \texttt{objeto} \in \mathcal{S}^f_{\wp(\texttt{CL})} \\ \texttt{nil} & \text{en otro caso} \end{cases}$$

Se define el objeto funcional `paradoja` del siguiente modo

```
(setq paradoja
      #'(lambda (objeto)
           (if (and (functionp objeto) (funcall inv? objeto))
               (not (funcall  objeto  objeto))
                nil)))
```

El objeto funcional `paradoja` está bien definido ya que antes de evaluar la expresión `(funcall objeto objeto)` se ha comprobado que `objeto` es un predicado Lisp. Notar que `paradoja` está en $\mathcal{S}^f_{\wp(\texttt{CL})}$ (es un algoritmo ya que termina para cualquier objeto Lisp), es decir que `(funcall inv? paradoja) = t`. Pero entonces, `(funcall paradoja paradoja) = (not (funcall paradoja paradoja))`, lo que lleva a una contradicción con la semántica de la primitiva Lisp `not`, contradicción que proviene de suponer que existe el objeto `inv?`.

■

Podemos interpretar, de un modo más informal, el objeto `paradoja` del siguiente modo. Como `paradoja` es una función con un argumento que devuelve `t` o `nil` y además la evaluación de sus llamadas siempre termina, podemos interpretarlo como la función característica de un conjunto, al que llamamos conjunto `paradoja` (los objetos que pertenecen al conjunto `paradoja` son aquellos sobre los que la llamada `(funcall paradoja objeto)` devuelve `t`). Los elementos de `paradoja` deben, por un lado, estar en el soporte, es decir ser algoritmos y, por otro, la evaluación de la llamada anterior debe ser `t`, por lo que `(funcall objeto objeto)` debe devolver `nil`, o, lo que es lo mismo, `objeto` no pertenece a sí mismo. De ahí que podamos interpretar `paradoja` como el conjunto de los conjuntos que no pertenecen a sí mismos.

Acabamos de mostrar que el conjunto de predicados Lisp es indecidible. Hay otras formas de interpretar el resultado anterior. El hecho de que en Common Lisp todo objeto tenga interpretación booleana hace que el resultado anterior también pueda leerse como que el conjunto de algoritmos Lisp es indecidible, es decir, dado un objeto funcional no existe un algoritmo que decida si el objeto es o no un algoritmo, dicho de otra forma, si termina la evaluación de cualquier llamada a ese objeto funcional. Más formalmente este resultado es una versión Lisp debida a Sergeraert ([102], página 17) de una variante del Problema de la Parada, variante que puede encontrarse en [59] (páginas 213 y 215):

> "`No existe algoritmo que pueda decidir si un código termina su ejecución para cualquier entrada`"

La versión Common Lisp en el contexto de los TADs del resultado anterior nos permite deducir la indecidibilidad de otra de las representaciones funcionales que hemos dado.

Corolario 3.6.3 *La representación funcional $\mathcal{R}^f_{\mathcal{U}^{\mathcal{U}}}$ de las aplicaciones de $\mathcal{U}$ en $\mathcal{U}$ (dada en el ejemplo 3.4.4) es indecidible.*

Hasta ahora nos hemos ocupado de la parte de representación y lo que hemos visto es que el uso de la codificación funcional hace que obtengamos representaciones indecidibles. Lo hemos hecho para el conjunto de los conjuntos de objetos Lisp por simplicidad, pero más adelante apoyándonos en resultados que introducimos a continuación, lo veremos para conjuntos de naturales, a partir de los que se extenderá a conjuntos con cardinalidad infinita de objetos de cualquier tipo.

Como ya hemos comentado, la representación de los datos condiciona la implementación de los operadores. Nos ocupamos de este aspecto a continuación.

3.6.2 Imposibilidad de implementar operadores globales

Los operadores de un tipo abstracto de datos se pueden clasificar en dos grupos: operadores de construcción y operadores de acceso. Al primer grupo pertenecen aquellos operadores cuyo resultado es un dato del tipo que se está construyendo (incluyendo los pensados para modificar datos del tipo). Los del segundo grupo son operadores que dan información sobre datos del tipo que se pretende diseñar, devolviendo datos de género diferente del género principal. Al trabajar con un tipo de datos, en particular con cualquiera de sus álgebras, en cuyo modelo subyace un conjunto, distinguimos dos tipos de operadores de acceso: operadores que manipulan y proporcionan información acerca de los elementos del conjunto y que denominaremos *operadores locales* y, operadores que describen o aportan información referente a todo el conjunto, a los que denominamos *operadores globales*.

En este apartado vamos a ver que algunos operadores globales no pueden implementarse a través de representaciones funcionales.

Por ejemplo, si consideramos el TAD $\mathcal{T}_{\wp(\mathbb{N})}$ descrito en el ejemplo 3.4.10 para los conjuntos de naturales, más concretamente el álgebra $\wp(\mathbb{N})$ allí definida, el operador $pertenece?_{\wp(\mathbb{N})}$ es un operador local mientras que el resto de operadores son operadores globales.

Evidentemente, todos los operadores anteriores, ya sean de naturaleza local o global, pueden implementarse a través de representaciones por extensión, pudiendo aparecer en este caso problemas de parcialidad. Por ejemplo, la unión de dos conjuntos (ambos en el soporte de una representación, por ejemplo, la representación $\mathcal{R}_{\wp(\mathbb{N})}$) puede producir salidas de rango en el sentido de que el objeto que representa a la unión no pertenezca al soporte de la representación ($\mathcal{R}_{\wp(\mathbb{N})}$, en el ejemplo). Por lo que respecta a las representaciones funcionales, los operadores locales pueden implementarse. Por ejemplo, una implementación del operador $pertenece?_{\wp(\mathbb{N})}$ a través de la representación del álgebra $\wp(\mathbb{N})$ dada en el ejemplo 3.4.10 es el programa $\mathtt{p}_{pertenece?}$=($\mathtt{c}^{\mathtt{p}_{pertenece?}}$, `(satisfies naturalp)` $\times\mathcal{S}^f_{\wp(\mathbb{N})}$, `(satisfies naturalp)`, $\mathcal{D}^f_{\wp(\mathbb{N})}$, $\mathcal{U}$) siendo

```
c^p_pertenece? ≡#'(lambda (objeto conjunto)
                    (funcall conjunto objeto))
```

Algunos operadores globales pueden implementarse. Por ejemplo, una implementación de la operación $union_{\wp(\mathbb{N})}$ a través de esa misma representación funcional viene dado por el programa $\mathtt{p}_{union} = (\mathtt{c}^{\mathtt{p}_{union}}, \mathcal{S}^f_{\wp(\mathbb{N})} \times \mathcal{S}^f_{\wp(\mathbb{N})}, \mathcal{D}^f_{\wp(\mathbb{N})}, \mathcal{D}^f_{\wp(\mathbb{N})}, \mathcal{D}^f_{\wp(\mathbb{N})})$ siendo

```
c^p_union ≡#'(lambda (conjunto1 conjunto2)
               #'(lambda (objeto)
                    (or (funcall conjunto1 objeto)
                        (funcall conjunto2 objeto))))
```

Análogamente, usando `and`, se implementa el operador global $inters_{\wp(\mathbb{N})}$.

Sin embargo, como demostramos a continuación, hay operadores globales que no pueden implementarse a través de la representación funcional $\mathcal{R}^f_{\wp(\mathbb{N})}$. Estos resultados también pueden encontrarse en un contexto teórico sobre calculabilidad en [59] (página 189), por ejemplo.

Para demostrar la no calculabilidad de algunos operadores nos apoyamos en otra versión del problema de la parada, un resultado fundamental en Informática Teórica y en Fundamentos de las Matemáticas. Su enunciado general, que puede encontrarse en [70] (página 277), es el siguiente

> "No existe algoritmo tal que a partir de un código *c* y un dato *d*, decida si la ejecución de *c* sobre *d* termina o no"

Enunciamos a continuación una versión Lisp de este problema, versión debida a Sergeraert (véase [102], corolario 10).

Teorema 3.6.4 *No existe algoritmo* `verificador` *perteneciente a* $\mathcal{A}(\mathcal{U},\mathcal{U};\mathcal{U})$ *tal que* `(funcall verificador x d) = t` *si y solo si* `x` *es un objeto funcional y la evaluación de* `(funcall x d)` *termina.*

El teorema anterior se lee como sigue: no existe un algoritmo Lisp con dos argumentos que evaluado devuelva `t` si y solo si se verifica que el primer argumento es un objeto funcional y

que la evaluación del primer argumento sobre el segundo termina.

Una demostración de ese resultado se obtiene como consecuencia inmediata de la equivalencia entre las máquinas Turing y las máquinas Lisp (usando como intermediario el λ-cálculo). Incluimos otra demostración mucho más directa, sin salir del contexto Lisp, demostración que puede encontrarse en [102] (página 17).

Previamente introducimos la siguiente definición que nos será útil en la demostración.

Definición 3.6.5 *Un conjunto A se dice* efectivamente enumerable *si existe un algoritmo que define una función $f\colon \mathbb{N} \to A$ sobreyectiva.*

Trasladando la definición anterior al marco Common Lisp se tiene que un tipo `A` es *efectivamente enumerable* si existe un algoritmo `x` de `(satisfies naturalp)` en $\mathcal{U}$ que define una función sobreyectiva.

Demostración: (del teorema 3.6.4)

La propiedad que utilizamos es que el Universo Lisp $\mathcal{U}$ es un tipo efectivamente enumerable, lo que significa que existe un algoritmo Lisp que denominamos `enumerador` del tipo `(satisfies naturalp)` en $\mathcal{U}$ que define una función sobreyectiva. La construcción del algoritmo `enumerador` reposa en el siguiente argumento: dado que los tipos predefinidos de objetos Lisp son un número finito (símbolos, números, listas, vectores, expresiones lambda, etc.) y dado que en cada tipo hay un conjunto numerable de objetos Lisp que podemos ordenar (por ejemplo, lexicográficamente), podemos definir un algoritmo que enumera a todos los objetos Lisp siguiendo las "diagonales del cuadrado" que forman tipos y objetos. El algoritmo `enumerador` es un algoritmo teórico, su código nos es indiferente, lo importante es que se puede escribir, aunque para ello haya que escribir un nuevo intérprete Common Lisp (en Common Lisp, por supuesto).

Ahora suponemos que existe el algoritmo `verificador` al que hace referencia el enunciado del teorema y apoyándonos en el algoritmo `enumerador` definimos el siguiente objeto funcional

```
(setq auxiliar
     #'(lambda (x)
        #'(lambda (objeto)
             (do ((n 0 (+ 1 n)))
                 ((not (funcall verificador x (funcall enumerador n))) n )))))
```

`(funcall auxiliar x)` es un objeto funcional que no termina su ejecución si y solo si para todo `n` de tipo `(satisfies naturalp)`, la evaluación de `(not (funcall verificador x (funcall enumerador n)))` devuelve `nil`; o, equivalentemente, si `x` es un objeto funcional y para todo `n` de tipo `(satisfies naturalp)` se tiene que `(funcall x (funcall enumerador n))` termina. Así, `(funcall auxiliar x)` es un objeto funcional que no termina su ejecución si y solo si `x` es un objeto funcional cuyas llamadas terminan para cualquier objeto Lisp.

Apoyándonos en la función anterior definimos el siguiente objeto funcional

```
(setq parada?
     #'(lambda (x)
          (if (not (functionp x))
              nil
             (not (funcall verificador (funcall auxiliar x) nil )))))
```

Si `x` es un objeto Lisp cualquiera, la evaluación de `(funcall parada?  x)` devuelve `nil` si y solo si, por definición de `parada?`, `(funcall verificador (funcall auxiliar x) nil)`

devuelve `t`, lo que por definición de `verificador` es equivalente a que `(funcall auxiliar x)` sea un objeto funcional y que la evaluación de `(funcall (funcall auxiliar x) nil)` termine. Ahora bien, como `(funcall (funcall auxiliar x) nil)` termina, por definición del objeto funcional `(funcall auxiliar x)` se tiene que existe $\texttt{d} \in \mathcal{U}$ tal que `(funcall x d)` no termina.

Así, tenemos

$$\texttt{(funcall parada? x)} = \begin{cases} \texttt{nil} & \text{si } \texttt{x} \text{ no termina sobre algún elemento de } \mathcal{U} \\ \texttt{t} & \text{en caso contrario} \end{cases}$$

y habríamos resuelto la variante del problema de la parada recogida en el corolario 3.6.3, o lo que es lo mismo, la representación funcional de las aplicaciones de $\mathcal{U}$ en $\mathcal{U}$ sería decidible.

■

Observar que el objeto `nil` sobre el que se evalua la llamada a `verificador` en el objeto `parada?` realmente podría ser cualquier objeto Lisp.

La existencia del algoritmo `enumerador` de `(satisfies naturalp)` en $\mathcal{U}$ implica además que la calculabilidad se puede modelar trabajando solamente con los naturales (resultado clásico debido a Gödel), ya que permite asociar un número a cada objeto Lisp del siguiente modo:

```
(setq numeroGodel
     #'(lambda (objeto)
             (do ((n 0 (+ 1 n)))
                 ((eq objeto (funcall enumerador n)) n ))))
```

Observar que por ser `enumerador` sobreyectivo, el objeto funcional anterior es un algoritmo de $\mathcal{U}$ en $\mathcal{U}$, es decir, termina para todo objeto Lisp. El algoritmo anterior formaliza en Common Lisp la "aritmetización de la calculabilidad" y, en particular, permite demostrar el siguiente resultado.

Teorema 3.6.6 *El conjunto* $\mathcal{A}$(`(satisfies naturalp)`, `(satisfies naturalp)`) *es indecidible.*

Demostración:

Si fuera decidible, existiría un algoritmo Lisp de $\mathcal{U}$ en $\mathcal{U}$, que denotaremos `inv?`, y que sería el invariante del conjunto $\mathcal{A}$(`(satisfies naturalp)`, `(satisfies naturalp)`). A partir de `inv?` podríamos construir el siguiente objeto funcional

```
(setq parada?
     #'(lambda (objeto)
          (if (not (functionp objeto))
              nil
             (funcall inv?  #'(lambda (n)
                     (funcall numeroGodel
                          (funcall objeto (funcall enumerador n))))))
```

que resuelve el problema de la parada, lo que contradice el corolario 3.6.3.

■

A partir de esto, es claro que para cualquier conjunto A infinito, la representación funcional del conjunto de las partes de A es siempre indecidible.

Como aplicaciones del teorema de la parada vamos a ver que algunos de los operadores globales del TAD para los conjuntos no son calculables a través de representaciones funcionales. Por simplicidad, comenzamos viéndolo para conjuntos de objetos Lisp.

Teorema 3.6.7 *El operador vacio?*$_{\wp(\texttt{CL})}$ *es no calculable a través de la representación funcional* $\mathcal{R}^f_{\wp(\texttt{CL})}$ *para conjuntos de objetos Lisp.*

Demostración:

Por evitar detalles técnicos, partimos de un algoritmo que llamamos `en_menos_de_n?`, de dominio `function` $\times\, \mathcal{U} \times$ `(satisfies naturalp)` y codominio $\mathcal{U}$, que toma como argumentos: un objeto funcional `f`, un objeto cualquiera `d` y un número natural `n`, y que devuelve `t` si la evaluación de `(funcall f d)` ha terminado tras `n` o menos llamadas de funciones y `nil` en otro caso. (La existencia teórica de este algoritmo es clara: además se puede construir basándonos en las mismas ideas usadas por Chaitin en [25], donde construye un algoritmo `try`, páginas 75 y siguientes de [25], y, para ello, se ve obligado a construir un nuevo intérprete Lisp, como hemos sugerido en la demostración del teorema 3.6.4 respecto de la existencia de `enumerador`.)

A partir de este algoritmo vamos a definir otro que denominamos `conjuntoAsociadoA` con dos argumentos, `f` y `d`, y que podemos interpretarlo como un constructor de conjuntos de objetos Lisp

```
(setq conjuntoAsociadoA
     #'(lambda (f d)
          #'(lambda (objeto)
               (if (not (and (integerp objeto) (>= objeto 0)))
                    nil
                   (funcall en_menos_de_n? f d objeto))))
```

Ahora, para cada objeto funcional `f` y para cada objeto Lisp `d`, `(funcall conjuntoAsociadoA f d)` define, según la representación funcional, un conjunto, ya que es un objeto funcional con un argumento cuya evaluación termina siempre, por lo que puede verse como la función característica de un conjunto. El conjunto al que representa está formado por aquellos objetos `x` sobre los que la evaluación de `(funcall (funcall conjuntoAsociadoA f d) x)` devuelve `t`. Estos objetos deben ser naturales y deben verificar que la llamada a `en_menos_de_n?` con `f` y `d` termina en menos de `objeto` llamadas. Así, `(funcall conjuntoAsociadoA f d)` es un conjunto formado por aquellos `n` para los que `f` termina sobre `d` en menos de `n` llamadas. Además, notar que se trata de un intervalo infinito de naturales ya que si un natural está en el conjunto, todos los mayores que él también lo están. Por tanto, el conjunto es vacío si `f` no termina al trabajar sobre `d` y, en cualquier otro caso es infinito.

Si el operador *vacio?*$_{\wp(\texttt{CL})}$ fuese calculable, utilizando el objeto anterior podríamos construir el objeto `verificador`:

```
(setq verificador
     #'(lambda (f d)
          (if (not (functionp f))
               nil
              (not (funcall vacio?  (funcall conjuntoAsociadoA f d))))))
```

siendo `vacio?` el código del programa que implementa a $vacio?_{\wp(\texttt{CL})}$ a través de la representación funcional $\mathcal{R}^f_{\wp(\texttt{CL})}$. El objeto `verificador` construido devuelve `t` sobre `f` y `d` si el conjunto `(funcall conjuntoAsociadoA f d)` no es vacío, lo que ocurre cuando `f` termina sobre `d`. Esto contradice el Teorema de la Parada, lo que demuestra que el operador $vacio?_{\wp(\texttt{CL})}$ es no calculable a través de la representación funcional $\mathcal{R}^f_{\wp(\texttt{CL})}$.

■

Con la misma idea de la demostración del teorema anterior podemos obtener un resultado análogo para otra representación.

Corolario 3.6.8 *El operador* $vacio?_{\wp(\mathbb{N})}$ *es no calculable a través de la representación funcional* $\mathcal{R}^f_{\wp(\mathbb{N})}$ *para conjuntos de números naturales.*

Demostración:

La idea general de la demostración es la misma que la del teorema anterior, varía en definir un objeto funcional que en lugar de representar a un conjunto genérico represente a un conjunto de naturales. Ese objeto funcional puede ser el siguiente:

```
(setq conjuntoAsociadoA
     #'(lambda (f d)
          #'(lambda (n)
               (funcall en_menos_de_n? f d n))))
```

A partir de él, se repite la última demostración.

■

En los últimos resultados hemos estudiado la propiedad de que un conjunto sea o no vacío. Es decir, hemos abordado el estudio de un problema definido por una propiedad. Consideramos el siguiente objeto funcional:

```
(setq noVacio?
     #'(lambda (conjunto)
          (do ((n 0 (+ 1 n)))
              ((funcall conjunto (funcall enumerador n)) t))))
```

El objeto funcional anterior devuelve `t` si `conjunto` no es vacío, por lo que podríamos pensar que el objeto anterior nos resuelve el problema. Sin embargo no es así ya que este objeto no es un algoritmo puesto que si `conjunto` es vacío, la evaluación de la llamada no termina nunca. Por tanto, no hemos resuelto el problema (algo ya sabido por el teorema anterior). Sin embargo, pese a no ser un algoritmo, el objeto funcional anterior detecta si `conjunto` posee la propiedad de no ser vacío aunque no detecta si posee o no la propiedad de ser vacío. Esto muestra que hay ocasiones en las que un problema definido por una propiedad no puede resolverse totalmente pero sí en parte y esto nos lleva a introducir las siguientes definiciones.

Definición 3.6.9

i) *Un problema definido por una propiedad es* decidible *si existe un algoritmo que decide si se tiene o no la propiedad.*

ii) *Un problema definido por una propiedad es* recursivamente enumerable *si existe un código que devuelve* `t` *si se tiene la propiedad.*

iii) *Un problema definido por una propiedad es* co-recursivamente enumerable *si existe un código que devuelve* `nil` *si no se tiene la propiedad.*

Observar que en la definición de recursivamente enumerable no se dice nada respecto del comportamiento del código cuando no se tiene la propiedad, podría incluso no terminar. Lo mismo ocurre en el caso co-recursivamente enumerable cuando se tiene la propiedad. Según estas definiciones y teniendo en cuenta el objeto funcional `noVacio?`, el problema de saber si un conjunto es *no vacío* es recursivamente enumerable. Equivalentemente, el problema de saber si un conjunto es *vacío* es co-recursivamente enumerable.

El siguiente resultado es bien conocido en Teoría de la Calculabilidad [70].

Teorema 3.6.10 *Un problema es decidible si y solo si es recursivamente enumerable y co-recursivamente enumerable.*

Volviendo a la implementación de operadores del TAD de conjuntos, como consecuencia de la no calculabilidad del operador *vacio*? podemos demostrar que otros operadores de naturaleza global tampoco son calculables. En primer lugar, nos ocupamos del operador que calcula el cardinal de un conjunto. Su comportamiento se describe a continuación (al usar codificación funcional pueden manejarse conjuntos infinitos por lo que este caso debe recogerse al definir el operador).

$$cardinal_{\wp(\texttt{CL})}(\texttt{C}) := \begin{cases} \texttt{INFINITO} & \text{si} \quad \alpha^f_{\wp(\texttt{CL})}(\texttt{C}) \text{ es infinito} \\ \texttt{n} & \text{si} \quad \alpha^f_{\wp(\texttt{CL})}(\texttt{C}) \text{ tiene } n \text{ elementos} \end{cases}$$

En la anterior definición `INFINITO` no es más que un símbolo Lisp empleado como etiqueta.

Corolario 3.6.11 *El operador* $cardinal_{\wp(\texttt{CL})}$ *es no calculable a través de la representación funcional* $\mathcal{R}^f_{\wp(\texttt{CL})}$ *para conjuntos de objetos Lisp.*

Demostración:

Si fuera calculable, existiría el algoritmo `cardinal` y podría construirse el siguiente algoritmo, que contradice al último teorema

```
(setq vacio?
     #'(lambda (conjunto)
          (let ((resp (funcall cardinal conjunto))
            (if (eq 'INFINITO resp)
                nil
                (zerop resp))))))
```

■

Corolario 3.6.12 *El operador* $conjuntos\_iguales?_{\wp(\texttt{CL})}$ *es no calculable en la representación funcional* $\mathcal{R}^f_{\wp(\texttt{CL})}$ *para conjuntos de objetos Lisp.*

Demostración:

Dado `conjuntos_iguales?`, el siguiente algoritmo serviría para implementar el operador `vacio?`

```
(setq vacio?
     #'(lambda (conjunto)
          (funcall conjuntos_iguales? conjunto #'(lambda (objeto) nil))))
```

lo que llevaría a contradicción.

■

Corolario 3.6.13 *El operador* $finito?_{\wp(\texttt{CL})}$ *es no calculable en la representación funcional* $\mathcal{R}^{f}_{\wp(\texttt{CL})}$ *para conjuntos de objetos Lisp.*

Demostración:

Nos apoyamos en la función `conjuntoAsociadoA` definida en la demostración del teorema 3.6.7. Como ya se indicó al definirla, los conjuntos construidos mediante `(funcall conjuntoAsociadoA f d)` tienen la propiedad de que son vacíos si y solo si son finitos. Así, si $finito?_{\wp(\texttt{CL})}$ fuera calculable, podríamos determinar si `(funcall conjuntoAsociadoA f d)` es vacío mediante

```
(setq vacioConjuntoAsociado?
     #'(lambda (f d)
          (funcall finito?  (funcall conjuntoAsociadoA f d)))
```

lo que, a su vez, nos permitiría implementar `verificador`

```
(setq verificador
     #'(lambda (f d)
          (if (not (functionp f))
               nil
              (not (funcall vacioConjuntoAsociado?  f d)))))
```

y nos llevaría a una contradicción como en la demostración del teorema 3.6.11.

■

Así, la representación funcional amplía el rango de conjuntos representables, pudiendo representar conjuntos de cardinalidad infinita, pero tiene sus contrapartidas negativas desde el punto de vista de la calculabilidad.

3.7 Especificando Implementaciones

En secciones anteriores hemos introducido nuestra noción de implementación de un tipo abstracto de datos. Esta noción va a ser clave en lo que sigue ya que, apoyándonos en ella, vamos a construir un marco formal, una categoría, sobre la que reposará nuestro análisis de las estructuras de datos utilizadas en EAT. Así, en esta sección, estudiaremos algunas implementaciones íntimamente relacionadas con las usadas en dicho software, que demostraremos poseen buenas propiedades desde el punto de vista de la Teoría de Categorías. Con esto justificaremos nuestra afirmación de que el patrón de implementación usado en EAT da lugar a las estructuras más

generales posibles para el tipo de cálculos que el programa debe realizar: permiten trabajar con cualquier representación de cualquier modelo involucrado en el cálculo.

Un proceso característico de Cálculo Simbólico en Topología Algebraica es el cálculo de un cierto invariante (de tipo algebraico) de un espacio topológico, para lo cual es habitual disponer de una versión de tipo combinatorio de dicho espacio (complejo simplicial, CW-complejo, etc.). Pues bien, es bastante habitual en tales procesos de cálculo que sea necesario construir algunas estructuras algebraicas intermedias que pueden responder a signaturas distintas de las álgebras de partida. Desde la perspectiva del programa, se parte de un objeto, que se entiende es la representación en la máquina de una cierta estructura algebraica, y durante el proceso de cálculo se construyen y se manipulan repesentaciones de otras álgebras distintas.

La forma general en la que en EAT se representa en la máquina un álgebra es algo tan simple como un conjunto de códigos funcionales. (La mínima información que debe disponerse para representar un álgebra son sus operaciones, en este caso, códigos que forman parte de programas que implementan a las operaciones.) Ahora bien, estos códigos por sí solos no portan toda la información que se necesita, deben ir acompañados de cierta información adicional que permita dar significado a la parte física (lo realmente almacenado en el computador) como representación de un álgebra concreta, información que no se encuentra en la máquina. Podemos decir que hay una parte que pertenece al universo de la máquina y otra al universo matemático. Como ya sabemos, la noción de implementación recoge estas dos componentes: contiene aquello que se pretende representar (visto como álgebra para una signatura) y contiene también la codificación en la máquina de esa estructura. Además, el álgebra y su representación en la máquina no forman piezas totalmente independientes, estando la relación existente entre ambas contenida también en la implementación, concretamente en las representaciones de los conjuntos soporte y en los programas que implementan a las operaciones: cada representación da una codificación de un conjunto soporte del álgebra a través de la función de abstracción; y, cada programa opera con las representaciones de los elementos del álgebra simulando el comportamiento de las operaciones (las condiciones impuestas en la definición de implementación vienen a establecer que haya coherencia entre el comportamiento de la parte física y la semántica de la parte matemática de la implementación). Así, por el tipo de información que recoge, la noción de implementación nos va a servir en nuestro propósito de identificar las propiedades que caracterizan a las estructuras de datos que se utilizan en EAT.

Como hemos comentado, los programas de cálculo de EAT deben poder crear, en tiempo de ejecución, representaciones de álgebras (por ejemplo, en algunos métodos se requiere la construcción de un complejo de cadenas como estructura intermedia en el cálculo de la homología de un espacio). Para ello, implícita o explícitamente, se debe disponer de un tipo de datos cuyos ejemplares sean *partes físicas de implementaciones de álgebras* (representaciones en la máquina de complejos de cadenas en el ejemplo). A nivel de modelos esto corresponde a la operación $()_{imp}$ estudiada en el capítulo anterior. A nivel físico, este hecho va a poder ser formalizado de un modo similar, considerando un tipo de datos cuyos ítems sean implementaciones de álgebras del tipo de partida (su signatura vendrá definida por la operación $()_{imp}$). Así, una implementación de un tal modelo es una implementación de una clase de implementaciones, lo que podríamos denominar una "implementación al cuadrado".

En esta sección desarrollaremos el formalismo necesario en el que enmarcar todas estas piezas y llegaremos a probar que el patrón de implementación usado en EAT da lugar a implementaciones que podríamos denominar canónicas, en el sentido de ser universales (objetos finales) en adecuadas categorías. Como se verá, estas implementaciones mantienen una estrecha relación con los objetos finales estudiados en el capítulo anterior. Las propiedades de finalidad

se obtendrán, al igual que ocurría a nivel de modelos, al restringirnos a implementaciones con los mismos soportes y con los mismos dominios para los operadores.

3.7.1 Representando implementaciones

Como hemos comentado anteriormente, uno de los aspectos que caracterizan a algunos de los programas de cálculo del sistema EAT es la necesidad de construir, en tiempo de ejecución, datos que constituyen representaciones en la máquina de estructuras algebraicas. Por ejemplo, pensemos que uno de estos programas debe construir la representación en la máquina de un grupo. Pues bien, si consideramos el TAD $\texttt{Grupo} = \langle \texttt{GRP}, \mathcal{G} \rangle$ donde $\texttt{GRP}$ es la signatura de grupo y $\mathcal{G}$ es la categoría de grupos, ese supuesto programa debe manipular objetos cuyos datos son implementaciones de $\texttt{Grupo}$, es decir, implícita o explícitamente, habrá de manejar una implementación del TAD $\texttt{Grupo}_{imp}$, resultado de aplicar a $\texttt{Grupo}$ la operación $()_{imp}$ introducida en el capítulo anterior y que, como ya sabemos, es el tipo que modela a las familias de grupos. Desde el punto de vista de la programación esto corresponde a la capacidad de EAT para trabajar con cualquier representación de cualquier ejemplar de una misma estructura algebraica, lo que lleva a que esas representaciones sean tratadas como datos.

Así, en abstracto, partimos de un TAD $\mathcal{T} = \langle \Sigma, \mathcal{C}(\Sigma) \rangle$ y de su categoría de implementaciones $Imp(\mathcal{T})$. Observar que cada implementación define de modo natural una Σ-álgebra: si $\mathcal{I} = (\underline{\mathcal{R}}, (\mathtt{p}_\sigma^{\mathcal{I}})_{\sigma \in \Omega})$ es una implementación de $\mathcal{T}$ con G-tipo $\underline{\mathtt{T}} = (\mathtt{T}_g)_{g \in G}$, entonces $\left\langle (\mathtt{T}_g)_{g \in G}, \{F(\mathtt{p}_\sigma^{\mathcal{I}}) \colon \mathtt{T}_\omega \to \mathtt{T}_v, Def(F(\mathtt{p}_\sigma^{\mathcal{I}}))\}_{(\sigma \colon \omega \to v) \in \Sigma} \right\rangle$ es Σ-álgebra. El álgebra definida por una implementación no recoge fielmente la parte física de esa implementación ya que no recoge el conjunto de códigos, solo las funciones definidas por los códigos. Esto es natural ya que la noción de álgebra es una noción de las matemáticas y para definirla se necesitan funciones y no códigos.

A partir de un TAD $\mathcal{T} = \langle \Sigma, \mathcal{C}(\Sigma) \rangle$ definimos la categoría $\mathcal{C}Imp(\mathcal{T})$ como la subcategoría plena de $PAlg(\Sigma)$ con objetos las Σ-álgebras definidas a partir de las implementaciones de $\mathcal{T}$, es decir, con objetos las Σ-álgebras $A = \left\langle (\mathtt{T}_g)_{g \in G}, \{F(\mathtt{p}_{\sigma_A}) \colon \mathtt{T}_\omega \to \mathtt{T}_v, Def(F(\mathtt{p}_{\sigma_A}))\}_{(\sigma \colon \omega \to v) \in \Sigma} \right\rangle$ para las que existe una implementación $\mathcal{I} = (\underline{\mathcal{R}}, (\mathtt{p}_\sigma^{\mathcal{I}})_{\sigma \in \Omega})$ del TAD $\mathcal{T}$ con G-tipo $\underline{\mathtt{T}} = (\mathtt{T}_g)_{g \in G}$ y con funciones definidas por los programas las funciones de la Σ-álgebra. Observar que distintas implementaciones pueden dar lugar a la misma álgebra, ya que pueden diferenciarse en los otros elementos que no quedan recogidos en la noción de álgebra definida (soportes de las representaciones, funciones de abstracción, igualdades de representación y códigos de los programas). Esta categoría $\mathcal{C}Imp(\mathcal{T})$ es pues un modelo matemático adecuado para las implementaciones de $\mathcal{T}$. La relación entre la categoría de implementaciones de un TAD $Imp(\mathcal{T})$ y la categoría $\mathcal{C}Imp(\mathcal{T})$ puede entenderse como un olvido de la primera en la segunda, olvido que, según el último comentario, no es inyectivo.

Todos los conjuntos soporte de las álgebras de $\mathcal{C}Imp(\mathcal{T})$ están en el Universo $\mathcal{U}$ de objetos Lisp, por lo que consideraremos éste como universo algebraico. Esta elección es natural teniendo en cuenta que pretendemos representar la parte de las implementaciones que vive en la máquina, o lo que es lo mismo, en el Universo Lisp.

Para trabajar como datos con la parte física de las implementaciones debemos aplicar a la categoría $\mathcal{C}Imp(\mathcal{T})$ la operación $()_{imp}$ definida en el capítulo anterior. Con ello obtenemos una categoría que, abusando de la notación, denotaremos por $\mathcal{C}Imp(\mathcal{T}_{imp})$ y que viene dada como sigue

- sus objetos son las Σ_{imp}-álgebras $A = \langle \mathtt{T}_{imp_\Sigma}, (\mathtt{T}_g)_{g\in G}, \{imp\_\sigma_A\colon \mathtt{T}_{imp_\Sigma} \times \mathtt{T}_\omega \to \mathtt{T}_v, Def(imp\_\sigma_A)\}_{(\sigma\colon \omega\to v)\in\Sigma}\rangle$ tales que, para todo $d \in \mathtt{T}_{imp_\Sigma}$, la Σ-álgebra $A_d = \langle (\mathtt{T}_g)_{g\in G}, \{imp\_\sigma_A(d,-)\colon \mathtt{T}_\omega \to \mathtt{T}_v, Def(imp\_\sigma_A(d,-))\}_{(\sigma\colon \omega\to v)\in\Sigma}\rangle$ con dominios de definición $Def(imp\_\sigma_A(d,-)) = \{d_\omega \mid (d, d_\omega) \in Def(imp\_\sigma_A)\}$, es objeto de $\mathcal{C}Imp(\mathcal{T})$;
- si A y B son Σ_{imp}-álgebras de $\mathcal{C}Imp(\mathcal{T}_{imp})$, un Σ_{imp}-morfismo $f\colon A \to B$, $f = (f_{imp_\Sigma}, (f_g)_{g\in G})$, es un morfismo de $\mathcal{C}Imp(\mathcal{T}_{imp})$ si, para todo $a \in A_{imp_\Sigma}$, $(f_g)_{g\in G}\colon A_a \to B_{f_{imp_\Sigma}(a)}$ es un morfismo de $\mathcal{C}Imp(\mathcal{T})$.

Usando la relación entre la operación $()_{imp}$ y la operación $()_{set}$ dada en el primer capítulo de esta memoria, podemos entender que cada objeto de $\mathcal{C}Imp(\mathcal{T}_{imp})$ representa a una familia de implementaciones del TAD $\mathcal{T}$, todas ellas con el mismo G-tipo, y estando presente en cada objeto solo lo que podríamos denominar parte física (máquina) de cada implementación de $\mathcal{T}$. Así, es natural denominar a la categoría $\mathcal{C}Imp(\mathcal{T}_{imp})$ *categoría de las familias de implementaciones de* $\mathcal{T}$.

Nuestro objetivo es demostrar la existencia de una familia de objetos de $\mathcal{C}Imp(\mathcal{T}_{imp})$, esto es, una "familia de familias de implementaciones", que en cierto sentido cubran a todos los objetos de $\mathcal{C}Imp(\mathcal{T}_{imp})$. Así, implementando esos objetos, se cubren todas las posibles implementaciones de $\mathcal{C}Imp(\mathcal{T})$. Al igual que hemos hecho en el capítulo anterior, descomponemos las categorías $\mathcal{C}Imp(\mathcal{T})$ y $\mathcal{C}Imp(\mathcal{T}_{imp})$ en subcategorías cuyas álgebras tienen soportes fijos para los géneros de la signatura de partida Σ. La razón para hacerlo así es agrupar implementaciones que trabajan sobre los mismos datos, es decir, implementaciones con los mismos dominios. Así, fijamos un G-conjunto $\underline{\mathtt{T}} = (\mathtt{T}_g)_{g\in G}$ donde, para cada género $g \in G$, $\mathtt{T}_g$ es un tipo concreto Common Lisp (esto es lo equivalente a fijar conjuntos al trabajar en cualquier universo algebraico, particularizado al caso de tomar como universo algebraico el de los objetos Lisp). Denotamos por $\mathcal{C}Imp^{\underline{\mathtt{T}}}(\mathcal{T})$ a la subcategoría plena de $\mathcal{C}Imp(\mathcal{T})$ con objetos los que poseen a $\underline{\mathtt{T}}$ como conjuntos soporte para los géneros, y denotamos por $\mathcal{C}Imp^{\underline{\mathtt{T}}}(\mathcal{T}_{imp})$ a la subcategoría plena de $\mathcal{C}Imp(\mathcal{T}_{imp})$ con objetos las Σ_{imp}-álgebras que tienen a los tipos fijados $\underline{\mathtt{T}}$ como conjuntos soporte para los géneros de Σ.

Para cada G-tipo $\underline{\mathtt{T}}$, la categoría $\mathcal{C}Imp^{\underline{\mathtt{T}}}(\mathcal{T}_{imp})$ posee un objeto que verifica que existe al menos un morfismo desde cualquier otro objeto en él. Denotamos por $\mathcal{M}^{\underline{\mathtt{T}}}$ a este objeto y una de sus descripciones es la siguiente:

- los soportes para los géneros de la signatura de partida Σ son los tipos $\underline{\mathtt{T}}$, luego, para cada $g \in G$, $\mathcal{M}^{\underline{\mathtt{T}}}_g = \mathtt{T}_g$;
- como soporte para el género distinguido $\mathcal{M}^{\underline{\mathtt{T}}}_{imp_\Sigma}$ se tiene el siguiente conjunto:

$$\mathcal{M}^{\underline{\mathtt{T}}}_{imp_\Sigma} = \Big\{(F(\mathtt{p}_\sigma))_{\sigma\in\Omega} \mid \exists\, \mathcal{I} = (\underline{\mathcal{R}}, (\mathtt{p}_\sigma)_{\sigma\in\Omega}) \in Obj(Imp(\mathcal{T})) \text{ tal que } \underline{\mathtt{T}} \text{ es el } G\text{-tipo de } \underline{\mathcal{R}}\Big\}$$

- para cada $\sigma\colon \omega \to v$ en Ω, la operación $imp\_\sigma_{\mathcal{M}^{\underline{\mathtt{T}}}}\colon \mathcal{M}^{\underline{\mathtt{T}}}_{imp_\Sigma} \times \mathtt{T}_\omega \to \mathtt{T}_v$ se define de manera natural por $imp\_\sigma_{\mathcal{M}^{\underline{\mathtt{T}}}}((F(\mathtt{p}_\eta))_{\eta\in\Omega}, \mathtt{d}_\omega) := F(\mathtt{p}_\sigma)(\mathtt{d}_\omega)$, siendo su dominio de definición

$$Def(imp\_\sigma_{\mathcal{M}^{\underline{\mathtt{T}}}}) = \Big\{\big((F(\mathtt{p}_\eta))_{\eta\in\Omega}, \mathtt{d}_\omega\big) \in \mathcal{M}^{\underline{\mathtt{T}}}_{imp_\Sigma} \times \mathtt{T}_\omega \mid d_\omega \in Def(F(\mathtt{p}_\sigma))\Big\}$$

Es claro que $\mathcal{M}^{\underline{\mathtt{T}}}$ es un objeto de $\mathcal{C}Imp^{\underline{\mathtt{T}}}(\mathcal{T}_{imp})$, ya que la Σ-álgebra que define cada tupla de funciones $(f_\sigma)_{\sigma\in\Omega} \in \mathcal{M}^{\underline{\mathtt{T}}}_{imp_\Sigma}$, es objeto de $\mathcal{C}Imp^{\underline{\mathtt{T}}}(\mathcal{T})$ (puesto que cualquier implementación de $\mathcal{T}$ que haga que la tupla $(f_\sigma)_{\sigma\in\Omega}$ esté en $\mathcal{M}^{\underline{\mathtt{T}}}_{imp_\Sigma}$, hace que la Σ-álgebra que define sea objeto de $\mathcal{C}Imp^{\underline{\mathtt{T}}}(\mathcal{T})$).

En [67] ya describimos el objeto anterior, aunque no analizamos sus propiedades en el sentido de la Teoría de Categorías, a lo que nos dedicamos a continuación.

Denotamos por $\mathcal{C}Imp^{\underline{\mathtt{T}},\{\}}(\mathcal{T})$ a la subcategoría de $\mathcal{C}Imp^{\underline{\mathtt{T}}}(\mathcal{T})$ con los mismos objetos y morfismos únicamente las identidades; y, por $\mathcal{C}Imp^{\underline{\mathtt{T}},\{\}}(\mathcal{T}_{imp})$ a la categoría obtenida aplicando la operación $()_{imp}$ a la anterior.

Podemos definir una inclusión canónica $I\colon \mathcal{C}Imp^{\underline{\mathtt{T}}}(\mathcal{T}) \to \mathcal{C}Imp^{\underline{\mathtt{T}}}(\mathcal{T}_{imp})$ que a cada Σ-álgebra $A = \langle \underline{\mathtt{T}}, \{f_\sigma, Def(f_\sigma)\}_{\sigma\in\Omega}\rangle$ de $\mathcal{C}Imp^{\underline{\mathtt{T}}}(\mathcal{T})$ le asocia la Σ_{imp}-álgebra $I(A) := \langle \mathtt{null}, \underline{\mathtt{T}}, \{imp\_\sigma_{I(A)}, Def(imp\_\sigma_{I(A)})\}_{\sigma\in\Omega}\rangle$, donde `null` es el tipo Common Lisp que tiene por único objeto a `nil` y, para cada $\sigma \in \Omega$, $Def(imp\_\sigma_{I(A)}) = \mathtt{null} \times Def(f_\sigma)$ y la función viene definida por $imp\_\sigma_{I(A)}(\mathtt{nil}, \mathtt{d}_\omega) := f_\sigma(\mathtt{d}_\omega)$.

La existencia de la inclusión I junto con la aplicación del teorema 2.8.10 a las categorías anteriores, permite obtener la siguiente propiedad del objeto $\mathcal{M}^{\underline{\mathtt{T}}}$

Teorema 3.7.1 *El objeto $\mathcal{M}^{\underline{\mathtt{T}}}$ es el coproducto en $\mathcal{C}Imp^{\underline{\mathtt{T}},\{\}}(\mathcal{T}_{imp})$ de los objetos de la imagen de la inclusión canónica de $\mathcal{C}Imp^{\underline{\mathtt{T}},\{\}}(\mathcal{T})$ en $\mathcal{C}Imp^{\underline{\mathtt{T}},\{\}}(\mathcal{T}_{imp})$.*

Otra propiedad de $\mathcal{M}^{\underline{\mathtt{T}}}$ queda recogida en el siguiente resultado, que es un caso particular del teorema 2.8.11.

Teorema 3.7.2 *La Σ_{imp}-álgebra $\mathcal{M}^{\underline{\mathtt{T}}}$ es final en la categoría $\mathcal{C}Imp^{\underline{\mathtt{T}},\{\}}(\mathcal{T}_{imp})$.*

El morfismo final tiene la siguiente expresión: si A es un objeto de $\mathcal{C}Imp^{\underline{\mathtt{T}},\{\}}(\mathcal{T}_{imp})$, el morfismo final $f = (f_{imp_\Sigma}, \underline{id})$ de A en $\mathcal{M}^{\underline{\mathtt{T}}}$ viene dado por la aplicación $f_{imp_\Sigma}\colon A_{imp_\Sigma} \to \mathcal{M}^{\underline{\mathtt{T}}}_{imp_\Sigma}$ que a cada $d \in A_{imp_\Sigma}$ le asigna la tupla de funciones $f_{imp_\Sigma}(d) := (imp\_\sigma_A(d, -))_{\sigma\in\Omega}$.

Si consideramos globalmente $\mathcal{C}Imp^{\{\}}(\mathcal{T}_{imp})$, subcategoría de $\mathcal{C}Imp(\mathcal{T}_{imp})$ que resulta de tomar solamente aquellos morfismos que son la identidad sobre los soportes de Σ, se tiene que la familia $\{\mathcal{C}Imp^{\underline{\mathtt{T}},\{\}}(\mathcal{T}_{imp})\}_{\underline{\mathtt{T}} \text{ es } G-\text{tipo}}$ define una partición de $\mathcal{C}Imp^{\{\}}(\mathcal{T}_{imp})$. Pues bien, usando el teorema anterior resulta sencillo probar el resultado recogido en el siguiente teorema.

Teorema 3.7.3 *La familia $\{\mathcal{M}^{\underline{\mathtt{T}}}\}_{\underline{\mathtt{T}} \text{ es } G-\text{tipo}}$ es final en la categoría $\mathcal{C}Imp^{\{\}}(\mathcal{T}_{imp})$.*

La existencia de un morfismo canónico de cada objeto de $\mathcal{C}Imp^{\underline{\mathtt{T}}}(\mathcal{T}_{imp})$ en $\mathcal{M}^{\underline{\mathtt{T}}}$ formaliza la idea intuitiva de que $\mathcal{M}^{\underline{\mathtt{T}}}$ recoge (representa) a *todas* las implementaciones del TAD inicial $\mathcal{T}$ con G-tipo $\underline{\mathtt{T}}$. Así, la familia final $\{\mathcal{M}^{\underline{\mathtt{T}}}\}_{\underline{\mathtt{T}} \text{ es } G-\text{tipo}}$ representa a todas las implementaciones del TAD de partida $\mathcal{T}$. Para poder trabajar con todas las implementaciones de $\mathcal{T}$ será suficiente considerar únicamente esta familia final. Denotaremos por $\mathcal{C}^{Imp}$ a la categoría cuyos objetos son las Σ_{imp}-álgebras $\mathcal{M}^{\underline{\mathtt{T}}}$ y con morfismos los existentes entre familias de implementaciones de $\mathcal{T}$ que son identidades sobre los géneros de Σ.

El punto importante es que, para cada TAD $\mathcal{T}$ y cada G-tipo $\underline{\mathtt{T}}$, la Σ_{imp}-álgebra $\mathcal{M}^{\underline{\mathtt{T}}}$ puede implementarse. Cualquier implementación de $\mathcal{M}^{\underline{\mathtt{T}}}$ posee como datos a las representaciones en la máquina de implementaciones de $\mathcal{T}$, siendo ésta la formalización de que en EAT se generan en tiempo de ejecución representaciones de estructuras algebraicas. Lo anterior hace que una

implementación de $\mathcal{M}^{\underline{\mathtt{T}}}$ pueda verse como una "implementación al cuadrado" o una "doble implementación" del TAD $\mathcal{T}$, ya que es una implementación cuyos datos son implementaciones de $\mathcal{T}$.

Damos a continuación ejemplos de implementaciones de dos Σ_{imp}-álgebras $\mathcal{M}^{\underline{\mathtt{T}}}$.

Ejemplo 3.7.4 Consideramos el TAD $\mathcal{T}_{\mathtt{GRP}} = \langle \mathtt{GRP}, \mathcal{C}(\mathtt{GRP}) \rangle$ siendo la signatura GRP la signatura de grupos,

$$\begin{array}{lcccl} prd & : & g & g & \rightarrow & g \\ inv & : & g & & \rightarrow & g \\ unt & : & & & \rightarrow & g \end{array}$$

y siendo $\mathcal{C}(\mathtt{GRP})$ la subcategoría plena de la categoría de grupos con objetos los grupos cociente de $\mathbb{Z}$ módulo n ($n > 1$), es decir, el conjunto de objetos es $\left\{\left\langle \mathbb{Z}/n\mathbb{Z}, \{+_{\mathbb{Z}/n\mathbb{Z}}, -_{\mathbb{Z}/n\mathbb{Z}}, 0_{\mathbb{Z}/n\mathbb{Z}}\}\right\rangle\right\}_{n\in\mathbb{N}, n>1}$ (cerrando por isomorfismo). Fijamos `integer` como tipo concreto para el único género de la signatura GRP y consideramos la $\mathtt{GRP}_{imp}$-álgebra $\mathcal{M}^{\mathtt{integer}}$. Una implementación de $\mathcal{M}^{\mathtt{integer}}$ en Common Lisp, y sin utilizar programación funcional, es la siguiente

- como representación para el género g de la signatura GRP tomamos la literal sobre `integer`;
- para el género distinguido $imp_{\mathtt{GRP}}$ tomamos la siguiente representación: como dominio `integer`; como soporte $\mathcal{S}_{imp_{\mathtt{GRP}}}$ los `n` que satisfacen el predicado `(and (naturalp n) (> n 1))`; como función de abstracción, $\alpha_{imp_{\mathtt{GRP}}}$: `integer` $\rightarrow \mathcal{M}^{\mathtt{integer}}_{imp_{\mathtt{GRP}}}$, definida, para cada `n` del soporte por $\alpha_{imp_{\mathtt{GRP}}}(\mathtt{n}) := (f^{\mathtt{n}}_{prd}, f^{\mathtt{n}}_{inv}, f^{\mathtt{n}}_{unt})$ siendo éstas últimas las funciones: $f^{\mathtt{n}}_{prd}$: `integer` $\times$ `integer` $\rightarrow$ `integer` función total definida por $f^{\mathtt{n}}_{prd}(\mathtt{x},\mathtt{y}) :=$ `(mod (+ x y) n)`, para todo `x`, `y` $\in$ `integer`; $f^{\mathtt{n}}_{inv}$: `integer` $\rightarrow$ `integer` también total y definida por $f^{\mathtt{n}}_{inv}(\mathtt{x}) :=$ `(mod (- x) n)`, para todo `x` $\in$ `integer`; y, finalmente la constante $f^{\mathtt{n}}_{unt}$ es la constante `0` $\in$ `integer`. Para cada `n` $\in \mathcal{S}_{imp_{\mathtt{GRP}}}$, la terna anterior es un elemento del conjunto $\mathcal{M}^{\mathtt{integer}}_{imp_{\mathtt{GRP}}}$. Una implementación de $\mathcal{T}_{\mathtt{GRP}}$ que nos permite asegurarlo es la formada por la representación que tiene a `integer` como dominio y como soporte, como función de abstracción la que a cada `m` de tipo `integer` lo lleva a su clase de equivalencia módulo n; y, como programas que implementan a las operaciones los siguientes
  - $\mathtt{p}^{\mathtt{n}}_{prd} = (\mathtt{c}^{\mathtt{p}^{\mathtt{n}}_{prd}}$, `integer` $\times$ `integer`, `integer`, `integer`, `integer`) siendo
    $\mathtt{c}^{\mathtt{p}^{\mathtt{n}}_{prd}} \equiv$ `#'(lambda (x y) (mod (+ x y) n))`
  - $\mathtt{p}^{\mathtt{n}}_{inv} = (\mathtt{c}^{\mathtt{p}^{\mathtt{n}}_{inv}}$, `integer`, `integer`, `integer`) siendo el código
    $\mathtt{c}^{\mathtt{p}^{\mathtt{n}}_{inv}} \equiv$ `#'(lambda (x) (mod (- x) n))`
  - $\mathtt{p}^{\mathtt{n}}_{unt} = (\mathtt{c}^{\mathtt{p}^{\mathtt{n}}_{unt}}$, `nil`, `nil`, `integer`) con código el siguiente objeto funcional
    $\mathtt{c}^{\mathtt{p}^{\mathtt{n}}_{unt}} \equiv$ `#'(lambda ( ) 0)`
- como programas que implementan a las operaciones consideramos los siguientes:
  - $\mathtt{p}_{imp\_prd} = (\mathtt{c}^{\mathtt{p}_{imp\_prd}}$, $\mathcal{S}_{imp_{\mathtt{GRP}}} \times$ `integer` $\times$ `integer`, `integer`, `integer`, `integer`, `integer`) siendo
    $\mathtt{c}^{\mathtt{p}_{imp\_prd}} \equiv$ `#'(lambda (n x y) (mod (+ x y) n))`

· $\mathtt{p}_{imp\_inv} = (\mathtt{c}^{\mathtt{p}_{imp\_inv}},\ \mathcal{S}_{imp_{\mathtt{GRP}}} \times$ `integer`, `integer`, `integer`, `integer`) siendo
$\mathtt{c}^{\mathtt{p}_{imp\_inv}} \equiv$ `#'(lambda (n x) (mod (- x) n))`

· $\mathtt{p}_{imp\_unt} = (\mathtt{c}^{\mathtt{p}_{imp\_unt}}, \mathcal{S}_{imp_{\mathtt{GRP}}},$ `integer`, `integer`) con
$\mathtt{c}^{\mathtt{p}_{imp\_unt}} \equiv$ `#'(lambda (n) 0)`

Acabamos de definir una implementación de una familia de implementaciones del TAD $\mathcal{T}_{\mathtt{GRP}}$.

□

Ejemplo 3.7.5 Recogemos a continuación un ejemplo concreto, próximo a EAT.

En el apartado 2.9.2 definimos el TAD $\mathcal{T}_{\mathtt{CS}} = \langle \mathtt{CS}, \mathcal{C}(\mathtt{CS}) \rangle$ que modela a los conjuntos simpliciales. La signatura `CS` consta de los géneros *nat* y *smp* y de las operaciones

$$\delta \quad : \quad nat \quad nat \quad smp \quad \rightarrow \quad smp$$

$$\eta \quad : \quad nat \quad nat \quad smp \quad \rightarrow \quad smp$$

Por su parte, la categoría $\mathcal{C}(\mathtt{CS})$ es una categoría equivalente a la categoría de los conjuntos simpliciales.

Es bien conocido que las esferas son espacios topológicos que pueden definirse como conjuntos simpliciales. Denotamos por $\mathtt{S}^m$ a la esfera de dimensión m que, como conjunto simplicial solo tiene dos símplices geométricos, uno en dimensión 0 y otro en dimensión m. Para justificar cómo EAT construye, en tiempo de ejecución, representaciones en la máquina de esferas (que puede necesitar como paso intermedio en alguno de sus cálculos), vamos a dar una implementación que recoja implementaciones de esferas, concretamente, una de cada dimensión. Estando en el nivel de implementación, debemos fijar tipos para los géneros *nat* y *smp*, tipos que proporcionarán un patrón sintáctico a los programas que trabajen con las implementaciones de las esferas. Para el género *nat*, consideramos el tipo `(satisfies naturalp)`. Para *smp*, nos apoyamos en lo desarrollado en el apartado 2.9.2 a nivel puramente algebraico. Ahí vimos cómo una forma adecuada de trabajar con símplices es partir de un conjunto graduado que, en cada dimensión, contenga los símplices geométricos del conjunto simplicial y, a partir de él, obtener un patrón que permita representar cualquier símplice (como abstracto). Como patrón general, un símplice abstracto puede representarse por un par formado por una lista estrictamente decreciente o vacía de enteros, que representan los operadores de degeneración que se aplican, y por un símplice geométrico. Esto nos lleva a considerar los datos del tipo

```
(defstruct asm dop gsm)
```

Fijamos como tipo $\mathtt{T}_{smp}$ el conjunto de datos del tipo `asm` anterior tales que el dato almacenado en el campo `dop` es una lista estrictamente decreciente o vacía de datos de tipo `integer` y el almacenado en el campo `gsm` es cualquier objeto del universo Lisp. Denotamos por $\langle(\mathtt{j_k} \ldots \mathtt{j_1}), \mathtt{a}\rangle$ al objeto `x` de $\mathtt{T}_{smp}$ tal que `(asm-dop x)` es $(\mathtt{j_k} \ldots \mathtt{j_1})$ y `(asm-gsm x)` es `a`.

Una vez fijados los tipos, podemos pasar a dar implementaciones concretas de esferas. Para ello, en este caso, consideramos el conjunto graduado $K_\cup = \{K_\cup^n\}_{n\in\mathbb{N}}$ siendo, para cada $n \in \mathbb{N}$, $K_\cup^n = \{(\mathtt{g}\ \mathtt{n})\}$ donde `g` es cualquier objeto de tipo `symbol` y `n` es de tipo `(satisfies naturalp)`. ($K_\cup$ es el paso natural a la máquina del conjunto graduado, denotado del mismo

modo y definido en la página 110, y utilizado en la parte algebraica de este ejemplo.) Así, los símplices geométricos de la esfera $\mathtt{S}^m$ los estamos representando por `(g 0)` y `(g m)`.

Denotamos por $\mathcal{R}^m$ a la representación de la esfera $\mathtt{S}^m$ con dominio $\mathtt{T}_{smp}$; como soporte el subconjunto formado por aquellos datos de $\mathtt{T}_{smp}$ tales que el dato almacenado en el campo `gsm` está en $K^n_\cup$ para algún $n \in \mathbb{N}$ y de forma que el mayor `integer` de la lista `dop` es menor que la longitud de la lista `dop` sumada con n. La función de abstracción $\alpha^m \colon \mathcal{D}^m \to \mathtt{S}^m$ viene definida por:

- $\alpha^m\big(\langle(\,),(\mathtt{g}\ 0)\rangle\big) := (\mathtt{g}\ 0)$
- $\alpha^m\big(\langle(\,),(\mathtt{g}\ \mathtt{n})\rangle\big) := \begin{cases} (\mathtt{g}\ \mathtt{m}) & \text{si } m = n \\ \eta_{n-1}\ldots\eta_0((\mathtt{g}\ 0)) & \text{si } m \neq n \end{cases}$
- $\alpha^m\big(\langle(\mathtt{j_k}\ldots\mathtt{j_1}),(\mathtt{g}\ \mathtt{n})\rangle\big) := \begin{cases} \eta_{j_k}\ldots\eta_{j_1}((\mathtt{g}\ \mathtt{m})) & \text{si } m = n \\ \eta_{n+k-1}\ldots\eta_0((\mathtt{g}\ 0)) & \text{si } m \neq n \end{cases}$

  donde η_j es la degeneración j-ésima en la dimensión adecuada.

Para completar la representación $\mathcal{R}^m$ de la esfera $\mathtt{S}^m$ hasta una implementación habrá que dar programas que implementen a las operaciones δ y η. Definimos los siguientes conjuntos, que usaremos como dominios de definición de los programas:

- $\mathtt{S}^\delta_\cup = \Big\{(\mathtt{i},\mathtt{n},\mathtt{x}) \mid \texttt{(<= 0 i)} \wedge \texttt{(<= i n)} \wedge \texttt{(> n 0)} \wedge \exists\ \mathtt{j} \in \texttt{(satisfies naturalp)}$ tal que $\texttt{(asm-gsm x)} \in K^{\mathtt{j}}_\cup \wedge \texttt{(= (+ (length (asm-dop x)) j) n)}\Big\}$
- $\mathtt{S}^\eta_\cup = \Big\{(\mathtt{i},\mathtt{n},\mathtt{x}) \mid \texttt{(<= 0 i)} \wedge \texttt{(<= i n)} \wedge \texttt{(>= n 0)} \wedge \exists\ \mathtt{j} \in \texttt{(satisfies naturalp)}$ tal que $\texttt{(asm-gsm x)} \in K^{\mathtt{j}}_\cup \wedge \texttt{(= (+ (length (asm-dop x)) j) n)}\Big\}$

Notar que la definición de los conjuntos $\mathtt{S}^\delta_\cup$ y $\mathtt{S}^\eta_\cup$ no depende de la dimensión de la esfera que estemos implementando.

Consideramos los programas $\mathtt{p}^m_\delta$= ($\mathtt{c}^m_\delta$, $\mathtt{S}^\delta_\cup$, `(satisfies naturalp)`, `(satisfies naturalp)`, $\mathtt{T}_{smp}$, $\mathtt{T}_{smp}$) y $\mathtt{p}^m_\eta$= ($\mathtt{c}^m_\eta$, $\mathtt{S}^\eta_\cup$, `(satisfies naturalp)`, `(satisfies naturalp)`, $\mathtt{T}_{smp}$, $\mathtt{T}_{smp}$) cuyos códigos definen las siguientes funciones:

- La correspondiente al operador cara viene dada, dependiendo de cómo sea el símplice, del siguiente modo
  - $F(\mathtt{c}^m_\delta)\big(\mathtt{i},\mathtt{n},\langle(\mathtt{j_{n-l}}\ldots\mathtt{j_1}),(\mathtt{g}\ \mathtt{l})\rangle\big) := \langle(\mathtt{j_{n-l}}-1\ldots\mathtt{j_h}-1\ldots\mathtt{j_1}),(\mathtt{g}\ \mathtt{l})\rangle$ si existe $\mathtt{h} \in \{1,\ldots,\mathtt{n}-\mathtt{l}\}$ tal que $\mathtt{j_h} = \mathtt{i}$ o $\mathtt{j_h} = \mathtt{i}-1$ siendo $\mathtt{j_h}+1 \neq \mathtt{i}$;
  - $F(\mathtt{c}^m_\delta)\big(\mathtt{i},\mathtt{n},\langle(\mathtt{j_{n-l}}\ldots\mathtt{j_1}),(\mathtt{g}\ \mathtt{l})\rangle\big) := \langle(\mathtt{n}-2\ldots 0),(\mathtt{g}\ 0)\rangle$, en otro caso.

  La función anterior se obtiene definiéndola sobre los símplices no degenerados como $F(\mathtt{c}^m_\delta)\big(\mathtt{i},\mathtt{n},\langle(\,),(\mathtt{g}\ \mathtt{n})\rangle\big) := \langle(\mathtt{n}-1\ldots 0),(\mathtt{g}\ 0)\rangle$, y extendiéndola a los degenerados mediante las propiedades exigidas a los conjuntos simpliciales como ya se explicó en este ejemplo desarrollado en el nivel algebraico.

· La función definida por el programa $\mathtt{p}_\eta^m$ que implementa al operador de degeneración es la función definida como $F(\mathtt{c}_\eta^m)\big(\mathtt{i},\mathtt{n},\langle(\mathtt{j_k}\ldots\mathtt{j_1}),\mathtt{x}\rangle\big) := \langle(\mathtt{j_k}+\mathtt{1}\ldots\mathtt{j_1}+\mathtt{1}\ \mathtt{i}\ \mathtt{j_{l-1}}\ldots\mathtt{j_1}),\mathtt{x}\rangle$ siendo l tal que $\mathtt{j_{l-1}} < \mathtt{i} \leq \mathtt{j_l}$.

Esta función viene de añadir al final de la lista de degeneraciones el índice correspondiente al operador de degeneración que se aplica, `i` en este caso, y de utilizar la propiedad $\eta_i^{n+1}\eta_j^n = \eta_{j+1}^{n+1}\eta_i^n$ si $i \leq j$, para conseguir que la lista de degeneraciones vuelva a ser decreciente.

Observar que todas las funciones anteriores son iguales, no dependen de m. Así, no hay problema en denotarlas como $f_\cup^\delta$ y $f_\cup^\eta$, respectivamente. Observar también que la expresión de esas funciones es totalmente constructiva, por lo que es claro que existen códigos que sobre los conjuntos $\mathtt{S}_\cup^\delta$ y $\mathtt{S}_\cup^\eta$ las definen, es decir, $f_\cup^\delta$ y $f_\cup^\eta$ son calculables.

Vamos a definir una implementación que recoja a todas las implementaciones anteriores (una por cada esfera). Consideramos la $\mathtt{CS}_{imp}$-álgebra que denotamos por $\mathcal{M}^{\underline{\mathtt{T}}}$, que es el objeto final de la categoría de $\mathtt{CS}_{imp}$-álgebras definidas a partir de implementaciones de $\mathcal{T}_{\mathtt{CS}}$ con $G_{\mathtt{CS}}$-tipo $\mathtt{T}_{smp}$ y `(satisfies naturalp)`. La $\mathtt{CS}_{imp}$-álgebra $\mathcal{M}^{\underline{\mathtt{T}}}$ resulta en este caso demasiado general. Como estamos trabajando solamente sobre esferas, y con un conjunto graduado fijo, podemos considerar la subálgebra de $\mathcal{M}^{\underline{\mathtt{T}}}$, que denotaremos por $\mathcal{M}^{K_\cup}$, definida a partir de implementaciones de esferas sobre $K_\cup$. En esta situación, en lugar de ir a una implementación de $\mathcal{M}^{K_\cup}$, preferimos mostrar otra implementación que se ciña un poco más a este modelo más pequeño, aunque sin perder de vista que también se trata de una implementación parcial del objeto $\mathcal{M}^{\underline{\mathtt{T}}}$. Denotamos por $\mathcal{I}^{K_\cup}$ a la implementación que definimos como sigue :

- tomamos la representación literal para los tipos $\mathtt{T}_{smp}$ y `(satisfies naturalp)`;
- para el género $imp_{\mathtt{CS}}$ tomamos la representación con dominio `(satisfies naturalp)`, soporte `(satisfies naturalp)`$\setminus\{0\}$ y función de abstracción $\alpha_{imp_{\mathtt{CS}}}^{K_\cup}$: (`satisfies naturalp`) $\to$ $\mathcal{M}_{imp_{\mathtt{CS}}}^{K_\cup}$ definida, para cada `m` del soporte por $\alpha_{imp_{\mathtt{CS}}}^{K_\cup}(\mathtt{m}) := (\delta^{\mathtt{m}},\eta^{\mathtt{m}})$ de forma que $\big\langle \mathtt{T}_{smp}$, `(satisfies naturalp)`, $\{\delta^{\mathtt{m}},\eta^{\mathtt{m}}\}\big\rangle$ represente a la esfera de dimensión m.

  Apoyándonos en las implementaciones anteriores, una para cada esfera $\mathtt{S}^m$, definimos las funciones $(\delta^{\mathtt{m}},\eta^{\mathtt{m}})$, imagen por la función de abstracción $\alpha_{imp_{\mathtt{CS}}}^{K_\cup}$ de `m`, del siguiente modo

  · $\delta^{\mathtt{m}}$: (`satisfies naturalp`)$\times$(`satisfies naturalp`)$\times\mathtt{T}_{smp}\to\mathtt{T}_{smp}$ es la función con dominio $\mathtt{S}_\cup^\delta$ y definida por $\delta^{\mathtt{m}} := f_\cup^\delta$

  · Análogamente, el dominio de la función $\eta^{\mathtt{m}}$: (`satisfies naturalp`) $\times$ (`satisfies naturalp`) $\times$ $\mathtt{T}_{smp}$ $\to$ $\mathtt{T}_{smp}$ es $\mathtt{S}_\cup^\eta$ y la función viene definida por $\eta^{\mathtt{m}} := f_\cup^\eta$
- como programas que implementan a las operaciones, consideramos $\mathtt{p}_{imp\_\delta}^{K_\cup}$= ($\mathtt{c}_{imp\_\delta}^{K_\cup}$, $\mathcal{S}_{imp_{\mathtt{CS}}}^{K_\cup}\times\mathtt{S}_\cup^\delta$, `(satisfies naturalp)`, `(satisfies naturalp)`, `(satisfies naturalp)`, $\mathtt{T}_{smp}$, $\mathtt{T}_{smp}$) siendo $\mathtt{c}_{imp\_\delta}^{K_\cup}$ un objeto funcional `#'(lambda (m i n x) ... )` que define la función $f_\cup^\delta$ (por lo que su comportamiento no dependerá del primer argumento). Análogamente para el programa $\mathtt{p}_{imp\_\eta}^{K_\cup}$.

Así, $\mathcal{I}^{K_\cup}$ es una implementación de la familia de implementaciones dadas anteriormente, una de cada esfera $\mathtt{S}^m$.

Podemos dar otra implementación $\widetilde{\mathcal{I}}^{K_\cup}$ de $\mathcal{M}^{K_\cup}$ modificando la función de abstracción. Así, definimos $\tilde{\alpha}^{K_\cup}_{imp_{\mathtt{CS}}}\colon (\mathtt{satisfies\ naturalp}) \to \mathcal{M}^{K_\cup}_{imp_{\mathtt{CS}}}$ como $\tilde{\alpha}^{K_\cup}_{imp_{\mathtt{CS}}}(\mathtt{m}) := (\tilde{\delta}^{\mathtt{m}}, \tilde{\eta}^{\mathtt{m}})$ siendo

- $\tilde{\delta}^{\mathtt{m}}\big(\mathtt{i}, \mathtt{n}, \langle(\ ), (\mathtt{g\ n})\rangle\big) := \begin{cases} \langle(\ ), (\mathtt{g\ n} - \mathtt{1})\rangle & \text{si } m \neq n \\ \langle(\mathtt{m} - \mathtt{1} \ldots \mathtt{0}), (\mathtt{g\ 0})\rangle & \text{si } m = n \end{cases}$

  extendiéndola a los símplices degenerados por medio de las propiedades de conjunto simplicial.

- $\tilde{\eta}^{\mathtt{m}}$ se define del mismo modo que en la implementación $\mathcal{I}^{K_\cup}$.

Para cada $\mathtt{m}$ del soporte de la representación, $\tilde{\alpha}^{K_\cup}_{imp_{\mathtt{CS}}}(\mathtt{m})$ está bien definida ya que existe una implementación de $\mathtt{S}^m$ con funciones $(\tilde{\delta}^{\mathtt{m}}, \tilde{\eta}^{\mathtt{m}})$. Basta tomar la misma representación $\mathcal{R}^m$ del caso anterior, los mismos soportes para los programas, pero variar el comportamiento del programa que implementa al operador δ. Es evidente que se puede dar un código para el programa que implementa al operador δ que sobre $\mathtt{S}^{\delta}_{\cup}$ defina la función $\tilde{\delta}^{\mathtt{m}}$ extendida a los símplices degenerados tal y como se ha explicado anteriormente.

Aunque $\mathcal{I}^{K_\cup}$ e $\widetilde{\mathcal{I}}^{K_\cup}$ son implementaciones diferentes, el único morfismo de cada una de ellas sobre el objeto final de la categoría de implementaciones llega a la misma tupla de funciones (se diferencian en los programas, pero no en las funciones definidas por ellos).

Partiendo de otro conjunto graduado, podemos definir otras implementaciones de $\mathtt{S}^m$. Consideramos el conjunto graduado $K_\sqcup = \{K^n_\sqcup\}_{n\in\mathbb{N}}$, siendo $K^0_\sqcup = \mathtt{(satisfies\ naturalp)}$ y $K^n_\sqcup = \{(\mathtt{g\ n})\}$ para cada $n > 0$ ($K_\sqcup$ es el trasladado a Common Lisp del conjunto graduado definido en la página 111 y denotado del mismo modo). Definimos una representación de cada esfera $\mathtt{S}^m$ con el mismo dominio que $\mathcal{R}^m$, es decir $\mathtt{T}_{smp}$, soporte análogo pero teniendo en cuenta que el dato almacenado en el campo `gsm` debe estar, en este caso, en $K^n_\sqcup$ para algún $n \in \mathbb{N}$, y definimos la función de abstracción $\alpha^m_\sqcup\colon \mathcal{D}^m \to \mathtt{S}^m$ como sigue

- $\alpha^m_\sqcup\big(\langle(\ ), \mathtt{n}\rangle\big) := \mathtt{n}$
- $\alpha^m_\sqcup\big(\langle(\ ), (\mathtt{g\ n})\rangle\big) := \begin{cases} (\mathtt{g\ m}) & \text{si } m = n \\ \eta_{n-1} \ldots \eta_0(\mathtt{m}) & \text{si } m \neq n \end{cases}$
- $\alpha^m_\sqcup\big(\langle(\mathtt{j_k} \ldots \mathtt{j_1}), (\mathtt{g\ n})\rangle\big) := \begin{cases} \eta_{j_k} \ldots \eta_{j_1}((\mathtt{g\ m})) & \text{si } m = n \\ \eta_{n+k-1} \ldots \eta_0(\mathtt{m}) & \text{si } m \neq n \end{cases}$

  donde η_j denota a la degeneración j-ésima en la dimensión adecuada.

Denotamos por $\mathtt{S}^{\delta}_\sqcup$ y $\mathtt{S}^{\eta}_\sqcup$ a los soportes de los programas, que se definen de modo análogo al caso anterior. Las funciones que deben definir estos programas son las siguientes

- La correspondiente al operador cara se define, sobre símplices no degenerados, como sigue
  - para cada $\mathtt{k} \in \mathtt{(satisfies\ naturalp)}$, $f^{\delta,m}_\sqcup\big(\mathtt{i}, \mathtt{n}, \langle(\mathtt{n} - \mathtt{1} \ldots \mathtt{0}), \mathtt{k}\rangle\big) := \langle(\mathtt{n} - \mathtt{2} \ldots \mathtt{0}), \mathtt{m}\rangle$;

· $f_{\sqcup}^{\delta,m}\big(\mathtt{i},\mathtt{n},\langle(\ ),(\mathtt{g}\ \mathtt{n})\rangle\big) := \langle(\mathtt{n}-\mathtt{1}\ldots\mathtt{0}),\mathtt{m}\rangle$;

y se extiende a los degenerados.

· La función definida por el programa que implementa al operador de degeneración actúa del siguiente modo:

$$f_{\sqcup}^{\eta,m}\big(\mathtt{i},\mathtt{n},\langle(\mathtt{j_k}\ldots\mathtt{j_1}),\mathtt{x}\rangle\big) := \langle(\mathtt{j_k}+\mathtt{1}\ldots\mathtt{j_1}+\mathtt{1}\ \mathtt{i}\ \mathtt{j_{l-1}}\ldots\mathtt{j_1}),\mathtt{x}\rangle$$

siendo l tal que $\mathtt{j_{l-1}} < \mathtt{i} \leq \mathtt{j_l}$.

Observar que, la función $f_{\sqcup}^{\eta,m}$ es la misma para cualquier $m \in \mathbb{N}$, por lo que la denotamos por $f_{\sqcup}^{\eta}$.

Las funciones $f_{\sqcup}^{\delta,m}$, para cada $m \in \mathbb{N}$, y $f_{\sqcup}^{\eta}$ son calculables y, por tanto, existirán códigos que las implementen. Damos una implementación $\mathcal{I}^{K_\sqcup}$ de la $\mathtt{CS}_{imp}$-álgebra $\mathcal{M}^{K_\sqcup}$, que indirectamente será implementación del álgebra final $\mathcal{M}^{\underline{\mathtt{T}}}$, considerando

- representación literal para $\mathtt{T}_{smp}$ y para `(satisfies naturalp)`;
- para el género $imp_{\mathtt{CS}}$, representación con dominio `(satisfies naturalp)`, soporte `(satisfies naturalp)`$\setminus\{0\}$ y función de abstracción $\alpha_{imp_{\mathtt{CS}}}^{K_\sqcup}$: (`satisfies naturalp`) $\to$ $\mathcal{M}_{imp_{\mathtt{CS}}}^{K_\sqcup}$ definida, para cada $\mathtt{m}$ del soporte, por $\alpha_{imp_{\mathtt{CS}}}^{K_\sqcup}(\mathtt{m}) := (f_{\sqcup}^{\delta,m}, f_{\sqcup}^{\eta})$.
- como programa que implementa a la operación que proviene del operador cara, consideramos el programa $\mathtt{p}_{imp\_\delta}^{K_\sqcup}$= ($\mathtt{c}_{imp\_\delta}^{K_\sqcup}$, $\mathcal{S}_{imp_{\mathtt{CS}}}^{K_\sqcup} \times \mathtt{S}_{\sqcup}^{\delta}$, `(satisfies naturalp)`, `(satisfies naturalp)`, `(satisfies naturalp)`, $\mathtt{T}_{smp}$, $\mathtt{T}_{smp}$), con $\mathtt{c}_{imp\_\delta}^{K_\sqcup}$ un objeto funcional `#'(lambda (m i n x) ... )` y siendo la función definida por el programa, la función $F(\mathtt{p}_{imp\_\delta}^{K_\sqcup})$: (`satisfies naturalp`) $\setminus\ \{0\} \times$ (`satisfies naturalp`) $\times$ (`satisfies naturalp`) $\times\ \mathtt{T}_{smp} \to \mathtt{T}_{smp}$, definida por $F(\mathtt{p}_{imp\_\delta}^{K_\sqcup})(\mathtt{m},\mathtt{i},\mathtt{n},\mathtt{x}) := f_{\sqcup}^{\delta,m}(\mathtt{i},\mathtt{n},\mathtt{x})$.
- como programa que implementa al operador de degeneración se considera $\mathtt{p}_{imp\_\eta}^{K_\sqcup}$= ($\mathtt{c}_{imp\_\eta}^{K_\sqcup}$, $\mathcal{S}_{imp_{\mathtt{CS}}}^{K_\sqcup} \times \mathtt{S}_{\sqcup}^{\eta}$, `(satisfies naturalp)`, `(satisfies naturalp)`, `(satisfies naturalp)`, $\mathtt{T}_{smp}$, $\mathtt{T}_{smp}$) siendo $\mathtt{c}_{imp\_\eta}^{K_\sqcup}$ un objeto funcional `#'(lambda (m i n x) ... )` de forma que la función que define el programa, $F(\mathtt{p}_{imp\_\eta}^{K_\sqcup})$, sea la función $f_{\sqcup}^{\eta}$.

Esta implementación, alcanza varios pares de funciones de los recogidos en el objeto final, uno para cada $m \in \mathbb{N}\setminus\{0\}$. Los pares de funciones alcanzados serán los formados por las funciones definidas por los programas de las implementaciones de las esferas que hemos considerado. Además, todos esos pares de funciones, poseen la misma función como segunda componente.

□

3.7.2 Implementaciones canónicas

En la sección anterior hemos definido a partir de un TAD $\mathcal{T}$ una familia de modelos $\{\mathcal{M}^{\underline{\mathtt{T}}}\}_{\underline{\mathtt{T}}\text{ es }G-\text{tipo}}$ de forma que los elementos de cada $\mathcal{M}^{\underline{\mathtt{T}}}$ corresponden a implementaciones de álgebras de $\mathcal{T}$. Según esto, disponer de implementaciones de los $\mathcal{M}^{\underline{\mathtt{T}}}$ nos permitirá poder manipular como datos la parte física de implementaciones de $\mathcal{T}$, justo lo que, como sabemos, se necesita en EAT. En esta sección se van a construir implementaciones de los modelos $\mathcal{M}^{\underline{\mathtt{T}}}$ que se

acercan en gran medida a las que en realidad fueron utilizadas en EAT. Además, se demostrará que el patrón de implementaciones usado en EAT (las estructuras de datos que emplean sus programas de cálculo) nos permite trabajar con la parte física de cualquier implementación de $\mathcal{T}$, lo que nos hace afirmar que dicho patrón es el más general entre las posibles formas de implementación de los modelos $\mathcal{M}^{\underline{\mathtt{T}}}$.

Observando la descripción de las Σ_{imp}-álgebras $\mathcal{M}^{\underline{\mathtt{T}}}$, parece natural usar programación funcional para implementarlas, aunque como muestran los dos ejemplos anteriores, la programación funcional no es estrictamente necesaria. Sin embargo, veremos cómo el uso de la programación funcional da lugar a implementaciones con propiedades de carácter universal en el sentido de la Teoría de Categorías, propiedades que no poseen, por ejemplo, las implementaciones a las que acabamos de referirnos.

Un álgebra consta, básicamente, de un conjunto de datos y unas operaciones que trabajan sobre ellos. Por tanto, si pensamos en la información necesaria para poder representar un álgebra, concluiremos que, como mínimo, esa información nos deberá permitir recuperar las operaciones del álgebra. Pensando en la representación en la máquina, como mínimo tendremos que disponer de códigos que implementen a las operaciones, códigos que deberán tomar sus datos en dominios fijados a priori (trabajar teniendo como datos representaciones de álgebras requiere fijar una base común para todas las representaciones: los datos que van a representar a los elementos de todas las álgebras; de ahí la partición según los G-tipos $\underline{\mathtt{T}}$).

Una observación importante es que hasta ahora nos hemos limitado a comentar la necesidad de generar en tiempo de ejecución representaciones de álgebras, pero además no debemos olvidar que hay casos en los que esas álgebras son de naturaleza infinita. El modo de representación que hemos denominado funcional, basado en el uso de la programación funcional, nos va a permitir trabajar con estructuras algebraicas de cualquier cardinalidad (es decir, finitas o numerables).

A continuación se estudia una implementación de $\mathcal{M}^{\underline{\mathtt{T}}}$, aunque en realidad lo que se va a obtener es toda una familia de implementaciones. (Aunque con menos detalles, esta familia de implementaciones ya la introdujimos en [67].)

Fijado un G-tipo $\underline{\mathtt{T}}$ y su correspondiente Σ_{imp}-álgebra $\mathcal{M}^{\underline{\mathtt{T}}}$, para representar las tuplas de funciones de $\mathcal{M}^{\underline{\mathtt{T}}}_{imp_\Sigma}$ tomaremos una estructura de almacenamiento, concretamente un tipo registro (`defstruct` en terminología Common Lisp), en cuyos campos se almacenarán objetos funcionales. No todos los registros formados por códigos funcionales son datos que interesan. Así, como soporte de la representación nos quedaremos solo con aquellos registros cuyos campos contienen códigos que provienen de implementaciones del TAD de partida $\mathcal{T}$. Con esta elección de soporte se plantea un problema para poder definir la función de abstracción. En efecto, los programas definen funciones pero no así los códigos, ya que éstos no contienen información sobre sus dominios de definición (los mismos objetos funcionales pueden formar parte de distintos programas y, por tanto, de distintas implementaciones; precisamente este hecho da lugar a la posibilidad de polimorfismo). Para poder definir la función de abstracción deberemos completar cada código hasta obtener un programa. Así, debemos asociar a cada código unos tipos de entrada y salida, esto no causa problema ya que que hemos fijado $\underline{\mathtt{T}}$, y un dominio de definición. Al quedar fijos los tipos de entrada, es claro que a cada código le podrá ser asignado un dominio de definición implícito, pero también podrá tomarse como dominio cualquiera de los subconjuntos de ese dominio implícito. Por tanto, no hay unicidad y nos vemos obligados a fijar de antemano los dominios de definición de los programas. Así, con $\underline{\mathtt{T}}$ fijo, para cada $\sigma\colon \omega \to v$ en Ω, denotamos por $\mathtt{S}^\sigma$ al subconjunto de $\mathtt{T}_\omega$ fijado como dominio de definición para el símbolo de operación σ. Denotamos por $\underline{\mathtt{S}}$ al correspondiente Ω-conjunto $(\mathtt{S}^\sigma)_{\sigma\in\Omega}$.

Utilizamos que estamos trabajando con signaturas que poseen un número finito de operaciones y suponemos que $(\sigma_1, \ldots, \sigma_m)$ es una enumeración de las operaciones de Σ (notación que emplearemos cuando sea necesaria una enumeración explícita; en otro caso, seguiremos manteniendo la que hemos utilizado hasta ahora: $(\sigma\colon \omega \to v)_{\sigma\in\Omega}$).

Fijados un G-tipo $\underline{\mathtt{T}}$ y un Ω-conjunto $\underline{\mathtt{S}}$, definimos la siguiente implementación de $\mathcal{M}^{\underline{\mathtt{T}}}$, a la que denotamos por $\mathcal{I}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}$:

- Para cada $g \in G$, tomamos la representación literal sobre $\mathtt{T}_g$.
- La representación $\mathcal{R}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma}$, para el soporte del género distinguido, $\mathcal{M}^{\underline{\mathtt{T}}}_{imp_\Sigma}$, tiene por dominio el conjunto de datos de una estructura con tantos campos como operaciones tiene la signatura de partida y tal que todos ellos son objetos funcionales. Formalizando lo anterior, tomamos como dominio $\mathcal{D}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma}$ los objetos Lisp del tipo `(satisfies D-can-p)`, donde para definir el predicado `D-can-p` utilizamos el tipo Lisp `IMP-g`

  `(defstruct IMP-g imp-`σ_1 `...` `imp-`σ_m`)`

  y siendo el predicado `D-can-p` el que decide si todos los campos de un dato del tipo anterior son o no objetos funcionales, es decir, el predicado `D-can-p` puede escribirse como sigue

```
(defun D-can-p (x)
     (and (typep x 'IMP-g)
          (functionp (IMP-g-imp-σ1 x))
                     ...
          (functionp (IMP-g-imp-σm x))))
```

  No todos los datos del tipo anterior interesan, sino solamente aquellos cuyos códigos, sobre los conjuntos $\underline{\mathtt{S}}$, definen programas que forman parte de alguna implementación del TAD de partida $\mathcal{T}$. Así consideramos el siguiente conjunto como soporte de la representación:

$$\mathcal{S}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma} = \Big\{ \mathtt{x} \in \texttt{(satisfies D-can-p)} \mid \exists\ \big(\underline{\mathcal{R}}, (\mathtt{p}_{\sigma_i})_{i=1}^m\big) \in Obj(Imp(\mathcal{T})) \text{ con } G\text{-tipo } \underline{\mathtt{T}},\ \Omega\text{-conjunto } \underline{\mathtt{S}} \text{ y tal que } \texttt{(eq c}^{\mathtt{p}_{\sigma_i}}\ \texttt{(IMP-g-imp-}\sigma_i\ \texttt{x))}, \text{ para todo } i = 1, \ldots, m \Big\}$$

  Observar que la igualdad que aparece en la definición del conjunto anterior es `eq`, la única razonable entre objetos funcionales.

  La función de abstracción $\alpha^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma}\colon \mathcal{S}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma} \to \mathcal{M}^{\underline{\mathtt{T}}}_{imp_\Sigma}$ lleva cada registro de objetos funcionales a la tupla de funciones, con dominios $\underline{\mathtt{S}}$, definidas por los códigos, es decir, para cada $\mathtt{x} \in \mathcal{S}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma}$, $\alpha^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma}(\mathtt{x}) := (F(\mathtt{p}_{\sigma_i}))_{i=1}^m$, siendo la tupla de funciones anterior la definida por los programas de cualquier implementación del TAD $\mathcal{T}$ con G-tipo $\underline{\mathtt{T}}$, Ω-conjunto $\underline{\mathtt{S}}$ y códigos de los programas los objetos funcionales almacenados en los campos de `x`. La primera observación necesaria en este punto es que la función de abstracción está bien definida, esto es, la tupla de funciones asociada a cada `x` del soporte por medio de la función de abstracción, no depende de la elección de la implementación del TAD de partida $\mathcal{T}$.

  Para terminar la definición de la representación $\mathcal{R}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma}$, queda por dar la igualdad de la representación. Tomamos como igualdad de representación $=_{\mathcal{R}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma}}$, la de la abstracción.

- Para concluir la definición de la implementación $\mathcal{I}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}$ construimos los programas que implementan a las operaciones de $\mathcal{M}^{\underline{\mathtt{T}}}$. Así, para cada operación $\sigma\colon \omega \to v$ en Ω, construimos un programa que implemente a la función $imp\_\sigma_{\mathcal{M}^{\underline{\mathtt{T}}}}\colon \mathcal{M}^{\underline{\mathtt{T}}}_{imp_\Sigma} \times \mathtt{T}_\omega \to \mathtt{T}_v$. Si $\omega = g_1 \dots g_n$, construimos el programa $\mathtt{p}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp\_\sigma} = (\mathtt{c}^{\mathtt{p}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp\_\sigma}}, \mathcal{S}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma} \times \mathtt{S}^\sigma,\ \mathcal{D}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma},\ \mathtt{T}_{g_1},\ \dots,\ \mathtt{T}_{g_n},\ \mathtt{T}_v)$, siendo el código asociado el siguiente objeto funcional

$$\mathtt{c}^{\mathtt{p}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp\_\sigma}} \equiv \texttt{\#'(lambda (x d1 ... dn)}\\ \texttt{(funcall (IMP-g-imp-}\sigma\ \texttt{x) d1 ... dn))}$$

En este punto consideramos conveniente hacer una observación sobre la elección de la igualdad de la abstracción como igualdad de la representación. Dados dos códigos (objetos funcionales), la igualdad intrínseca Common Lisp para ellos es la dada por `eq`, es decir, la igualdad como objetos de la máquina. Esta igualdad es muy débil, identifica pocos objetos, pero es la única natural si pensamos únicamente en los códigos. Ahora bien, cuando a un código le asociamos un dominio de definición, podemos identificarlo con la función que define y, en ese caso, lo natural es considerar como igualdad la igualdad matemática entre funciones. Esta relación de igualdad es la que habitualmente se denomina igualdad comportamental y que definimos a continuación.

Definición 3.7.6 *Sean* $\mathtt{c_1}$ *y* $\mathtt{c_2}$ *objetos funcionales y* $\mathtt{p_i} = (\mathtt{c_i}, \mathtt{S}, \mathtt{T}_1,\ \dots,\ \mathtt{T_n},\ \mathtt{T})$ *programa para* $i = 1, 2$. *Se dice que* $\mathtt{c_1}$ *y* $\mathtt{c_2}$ *son* iguales comportamentalmente (respecto a los tipos $\mathtt{T}_1,\ \dots,\ \mathtt{T_n},\ \mathtt{T}$ y el dominio $\mathtt{S}$) *si los programas* $\mathtt{p_1}$ *y* $\mathtt{p_2}$ *definen la misma función.*

La siguiente proposición asegura que lo que acabamos de definir es una implementación.

Proposición 3.7.7 $\mathcal{I}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}$ *es una implementación de la* Σ_{imp}*-álgebra* $\mathcal{M}^{\underline{\mathtt{T}}}$.

Demostración:

Sin incluir la demostración completa, sí que vamos a comentar algunos detalles de ésta que consideramos interesantes:

· Para cada $\sigma \in \Omega$, $\mathtt{p}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp\_\sigma}$ es un programa. El objeto funcional $\mathtt{c}^{\mathtt{p}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp\_\sigma}}$ puede aplicarse sobre los tipos $\mathcal{D}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma},\ \mathtt{T}_{g_1},\ \dots,\ \mathtt{T}_{g_n}$ puesto que lo único que se usa es el acceso a un campo de un dato de tipo `defstruct`, que debe ser un objeto funcional (lo es por definición del dominio $\mathcal{D}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma}$). Por su parte, $\mathtt{c}^{\mathtt{p}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp\_\sigma}}$ define un algoritmo de $\mathcal{S}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma} \times \mathtt{S}^\sigma$ en $\mathtt{T}_g$, puesto que para cada dato $(\mathtt{x},\ \mathtt{d}_1, \dots, \mathtt{d}_n)$ del soporte del programa, por ser $\mathtt{x}$ del soporte de la representación $\mathcal{R}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma}$, al menos existe una implementación de $\mathcal{T}$ con código `(IMP-g-imp-`σ `x)` y con dominio de definición $\mathtt{S}^\sigma$, luego la ejecución termina. Por último, observar que también se respetan las igualdades de los tipos concretos. En todo lo anterior se utiliza que la representación para cada tipo fijado es la literal.

· Para cada $\sigma \in \Omega$, el correspondiente $\mathtt{p}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp\_\sigma}$ es un programa compatible con las representaciones. El haber tomado la igualdad de la abstracción como igualdad de la representación para el género distinguido y las representaciones literales para el resto de géneros, hace que sea compatible con las igualdades de las representaciones. Por la misma razón, $\mathtt{p}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp\_\sigma}$ también es coherente con las representaciones.

· Para cada $\sigma \in \Omega$, el programa $\mathtt{p}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp\_\sigma}$ implementa al operador $imp\_\sigma_{\mathcal{M}^{\underline{\mathtt{T}}}}$. En la demostración basta con usar la descomposición del soporte del programa como producto del soporte de la representación para el género distinguido por el conjunto $\mathtt{S}^\sigma$ fijado, conocer el significado del operador de acceso a las componentes del `defstruct` y usar que se están manejando las representaciones literales para los géneros no distinguidos.

■

En los comentarios anteriores queda claro que la elección de las representaciones literales para los géneros de la signatura de partida es clave en la demostración. Pese a que esta elección pueda parecer restrictiva no lo es en absoluto, ya que podemos cubrir la representación de todas las implementaciones de $\mathcal{T}$. En general, una representación recoge la codificación en la máquina de un modelo, pero en este caso el modelo que se pretende representar está ya en la máquina y, por tanto, no hace falta considerar una función de abstracción que no sea la identidad (ni una igualdad de representación que no sea la del propio dominio). La base de todo ello es que estamos recogiendo la parte física de las implementaciones de un TAD $\mathcal{T}$, luego no hay un paso propio del modelo a la representación sino que el paso es el trivial en todos los géneros salvo en el que se agrupan.

Las implementaciones $\mathcal{I}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}$ verifican propiedades universales en el sentido de la Teoría de Categorías, que mostraremos en posteriores apartados. Por ello las denominamos *implementaciones canónicas*.

Hay dos aspectos técnicos de $\mathcal{I}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}$ que serán utilizados más adelante y que por ese motivo comentamos a continuación:

- El conjunto imagen por la función de abstracción $\alpha^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma}$ correspondiente al género distinguido, puede describirse del siguiente modo:

$$\alpha^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma}(\mathcal{S}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma}) = \left\{(f_\sigma)_{\sigma\in\Omega} \in \mathcal{M}^{\underline{\mathtt{T}}}_{imp_\Sigma} \;\mid\; Def(f_\sigma) = \mathtt{S}^\sigma, \forall\sigma \in \Omega\right\}$$

- Para cada $\sigma \in \Omega$, el soporte de cada programa $\mathtt{p}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp\_\sigma}$, programa que implementa al operador $imp\_\sigma_{\mathcal{M}^{\underline{\mathtt{T}}}}$, puede describirse como un rectángulo. En particular, factoriza en la componente correspondiente a imp_Σ y además lo hace por el soporte de la representación

$$\mathtt{S}^{\mathtt{p}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp\_\sigma}} = \mathcal{S}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma} \times \mathtt{S}^\sigma$$

Denotamos por $\mathcal{M}^{\underline{\mathtt{T}},\underline{\mathtt{S}}}_{imp_\Sigma}$ al conjunto imagen de la función de abstracción para el género distinguido. Así, $\mathcal{M}^{\underline{\mathtt{T}},\underline{\mathtt{S}}}_{imp_\Sigma}$ es el conjunto de tuplas de funciones que provienen de implementaciones de $\mathcal{T}$ con G-tipo $\underline{\mathtt{T}}$ y Ω-conjunto $\underline{\mathtt{S}}$. Fijado un G-tipo $\underline{\mathtt{T}}$, para cada Ω-conjunto $\underline{\mathtt{S}}$, denotamos por $\mathcal{M}^{\underline{\mathtt{T}},\underline{\mathtt{S}}}$ a la correspondiente subálgebra de $\mathcal{M}^{\underline{\mathtt{T}}}$. Para cada G-tipo $\underline{\mathtt{T}}$ se tiene una partición del conjunto $\mathcal{M}^{\underline{\mathtt{T}}}_{imp_\Sigma}$ determinada por los posibles dominios de definición de las funciones:

$$\mathcal{M}^{\underline{\mathtt{T}}}_{imp_\Sigma} = \bigcup_{(\mathtt{S}^\sigma\subseteq\mathtt{T}_\omega)_{(\sigma:\,\omega\to g)\in\Omega}} \mathcal{M}^{\underline{\mathtt{T}},\underline{\mathtt{S}}}_{imp_\Sigma}$$

La imagen de la función de abstracción $\alpha^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma}$ cae sobre (y cubre) $\mathcal{M}^{\underline{\mathtt{T}},\underline{\mathtt{S}}}_{imp_\Sigma}$, lo que nos permite pensar en $\mathcal{I}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}$ como implementación del álgebra $\mathcal{M}^{\underline{\mathtt{T}},\underline{\mathtt{S}}}$. La representación $\mathcal{R}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma}$ de $\mathcal{M}^{\underline{\mathtt{T}}}_{imp_\Sigma}$ es plena sobre $\mathcal{M}^{\underline{\mathtt{T}},\underline{\mathtt{S}}}_{imp_\Sigma}$.

3.7.3 Categorías de implementaciones

Nuestro siguiente objetivo será mostrar algunas propiedades categoriales que poseen las implementaciones $\mathcal{I}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}$, propiedades teóricas que tienen su interpretación en el proceso de cálculo que realizan los programas de EAT. En primer lugar, vamos a estudiar la categoría de implementaciones de la Σ_{imp}-álgebra $\mathcal{M}^{\underline{\mathtt{T}}}$. Al igual que ocurría en el caso algebraico, el resultado de universalidad de $\mathcal{I}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}$ no se va a dar en toda la categoría de implementaciones, sino en una subcategoría propia. Para definir esa subcategoría nos basamos en algunas de las propiedades que hemos destacado de las implementaciones canónicas.

Así, denotamos por $Imp^{\underline{\mathtt{T}},\underline{\mathtt{S}},\{\}}(\mathcal{M}^{\underline{\mathtt{T}}})$ a la subcategoría de la categoría de implementaciones de $\mathcal{M}^{\underline{\mathtt{T}}}$ con objetos las implementaciones que verifican las siguientes condiciones:

i) para cada $g \in G$, la representación para el tipo concreto $\mathtt{T}_g$ fijado, es la representación literal sobre $\mathtt{T}_g$, que denotamos por $\mathcal{R}^l_{\mathtt{T}_g}$;

ii) el conjunto imagen por la función de abstracción del género imp_Σ es subconjunto de $\mathcal{M}^{\underline{\mathtt{T}},\underline{\mathtt{S}}}_{imp_\Sigma}$;

iii) los dominios de definición de los programas que implementan a las operaciones son cubos de la forma: $\mathcal{S}_{imp_\Sigma} \times \mathtt{S}^\sigma$.

Como morfismos de la categoría $Imp^{\underline{\mathtt{T}},\underline{\mathtt{S}},\{\}}(\mathcal{M}^{\underline{\mathtt{T}}})$ se toman aquellas i–transformaciones que son la identidad sobre todos los modelos y sobre los soportes de todas las representaciones salvo la del género distinguido imp_Σ.

Es claro que cada implementación canónica $\mathcal{I}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}$ es objeto de la correspondiente categoría $Imp^{\underline{\mathtt{T}},\underline{\mathtt{S}},\{\}}(\mathcal{M}^{\underline{\mathtt{T}}})$.

Ya hemos comentado con anterioridad la adecuación del modelo $\mathcal{M}^{\underline{\mathtt{T}}}$ a las necesidades de los procesos de cálculo que se realizan en EAT y, en general, de cualquier sistema que requiera la manipulación de (partes físicas de) implementaciones de estructuras algebraicas. Por tanto, disponer de implementaciones de $\mathcal{M}^{\underline{\mathtt{T}}}$ nos va a permitir atender estos requerimientos. Fijados $\underline{\mathtt{T}}$ y $\underline{\mathtt{S}}$, veremos que la implementación $\mathcal{I}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}$ posee buenas propiedades en la categoría $Imp^{\underline{\mathtt{T}},\underline{\mathtt{S}},\{\}}(\mathcal{M}^{\underline{\mathtt{T}}})$. A continuación, explicamos brevemente las restricciones establecidas al definir esta categoría de implementaciones.

La condición *i*) impuesta a los objetos de $Imp^{\underline{\mathtt{T}},\underline{\mathtt{S}},\{\}}(\mathcal{M}^{\underline{\mathtt{T}}})$ es totalmente natural: una representación contiene el paso de un modelo, que es lo que se pretende representar, a un dominio, cuyos datos son los utilizados en la máquina; en nuestro caso el modelo ya está en la máquina, más aún, modelo y dominio coinciden, luego lo natural es que cada dato concreto, se represente a sí mismo y no lleve consigo un proceso de codificación distinto del trivial; así, el paso de abstracción es el trivial y de ahí el considerar las representaciones literales como representaciones para los tipos fijados. Respecto de la condición *ii*), estamos agrupando en la implementación de $\mathcal{M}^{\underline{\mathtt{T}}}$ la parte física de implementaciones que provienen de otro TAD; así, fijado un Ω-conjunto $\underline{\mathtt{S}}$, las que quedan recogidas deben provenir del TAD de partida y tener como dominios de definición de los programas los $\underline{\mathtt{S}}$ fijados. La condición *iii*) viene a imponer que todas las implementaciones recogidas en el dominio de la implementación de $\mathcal{M}^{\underline{\mathtt{T}}}$ tengan como dominios de definición de los programas al Ω-conjunto fijado. Como el soporte del género distinguido actúa simplemente como conjunto para indexar las implementaciones recogidas, es natural que los dominios de definición de los programas de esta implementación factoricen por la componente que indexa las implementaciones de partida y además que lo hagan de ese modo.

Las condiciones impuestas a los morfismos de la categoría $Imp^{\underline{\mathtt{T}},\underline{\mathtt{S}},\{\}}(\mathcal{M}^{\underline{\mathtt{T}}})$ también merecen cierta justificación. En la parte de los modelos, imponemos que todas las aplicaciones sean la identidad: en el caso de los géneros de partida está claro ya que los modelos son los tipos $\underline{\mathtt{T}}$, tipos que se representan a sí mismos (a través de las representaciones literales); en el caso del género distinguido, el modelo es, en todas las implementaciones de la categoría, el conjunto $\mathcal{M}^{\underline{\mathtt{T}}}_{imp_\Sigma}$ (aunque realmente la abstracción cae dentro de $\mathcal{M}^{\underline{\mathtt{T}},\underline{\mathtt{S}}}_{imp_\Sigma}$), conjunto formado por tuplas de funciones todas ellas con los mismos dominios, por lo que no parece natural incluir "endo-transformaciones". En la parte física de las i–transformaciones, es decir, en las transformaciones entre dominios, al tomar representaciones literales y las mismas en todas las implementaciones, es natural imponer que entre los tipos fijados no haya transformaciones, es decir, de algún modo estamos fijando la interpretación de cada dato del dominio en las distintas implementaciones del TAD $\mathcal{T}$.

Con los comentarios anteriores hemos querido aclarar las restricciones impuestas a la categoría de implementaciones de $\mathcal{M}^{\underline{\mathtt{T}}}$, lo que, mirando hacia atrás, justifica las restricciones impuestas a las categorías estudiadas en el caso algebraico en el capítulo anterior. De hecho, aquellas restricciones fueron inspiradas por nuestro interés inicial en los aspectos relacionados con la implementación.

Así, fijados un G-conjunto $\underline{\mathtt{T}}$ y un Ω-conjunto $\underline{\mathtt{S}}$ consideramos la categoría $Imp^{\underline{\mathtt{T}},\underline{\mathtt{S}},\{\}}(\mathcal{M}^{\underline{\mathtt{T}}})$ de implementaciones del modelo $\mathcal{M}^{\underline{\mathtt{T}}}$, que contienen las representaciones en la máquina de adecuadas implementaciones del tipo abstracto de partida $\mathcal{T}$. Cada implementación $\mathcal{I}$ de $\mathcal{T}$ con G-tipo $\underline{\mathtt{T}}$ y Ω-conjunto $\underline{\mathtt{S}}$ da lugar a una implementación $\mathcal{I}^*$ de $\mathcal{M}^{\underline{\mathtt{T}}}$. La definición de $\mathcal{I}^*$ es análoga a la definición de $\mathcal{I}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}$ tomando como soporte del género distinguido $\mathcal{S}^{\mathcal{I}^*,\underline{\mathtt{T}},\underline{\mathtt{S}}}_{imp_\Sigma}$, el conjunto unipuntual formado por la tupla de tipo `IMP-g` formada por los códigos de los programas de $\mathcal{I}$. Es fácil probar que $\mathcal{I}^*$ es una implementación de $\mathcal{M}^{\underline{\mathtt{T}}}$, aunque se trata de una implementación muy parcial. La función de abstracción sobre el género distinguido solo alcanza a un elemento del correspondiente dominio, a una única tupla de funciones. Por otro lado, conviene señalar que este paso de implementaciones de $\mathcal{T}$ a implementaciones de $\mathcal{M}^{\underline{\mathtt{T}}}$ no es inyectivo (pueden existir implementaciones de $\mathcal{T}$ con G-tipo $\underline{\mathtt{T}}$ y Ω-conjunto $\underline{\mathtt{S}}$, con los mismos objetos funcionales formando parte de los programas que implementan a los operadores y, sin embargo, diferir en los modelos, en las funciones de abstracción o en las igualdades). Si en la categoría de las implementaciones de $\mathcal{T}$ restringimos los morfismos a las identidades (nos quedamos con la categoría discreta), podemos ver la construcción anterior como un funtor inclusión de $Imp^{\underline{\mathtt{T}},\underline{\mathtt{S}},\{\}}(\mathcal{T})$ en $Imp^{\underline{\mathtt{T}},\underline{\mathtt{S}},\{\}}(\mathcal{M}^{\underline{\mathtt{T}}})$. Además, para cada implementación $\mathcal{I}$ de $\mathcal{T}$ con G-tipo $\underline{\mathtt{T}}$ y Ω-conjunto $\underline{\mathtt{S}}$, su correspondiente $\mathcal{I}^*$ puede ser pensada como una pieza unipuntual de $\mathcal{I}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}$. Todo esto queda recogido en el siguiente teorema:

Teorema 3.7.8 *La implementación $\mathcal{I}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}$ es el coproducto en $Imp^{\underline{\mathtt{T}},\underline{\mathtt{S}},\{\}}(\mathcal{M}^{\underline{\mathtt{T}}})$ de las implementaciones de la imagen del funtor inclusión de la categoría de implementaciones de $\mathcal{T}$, $Imp^{\underline{\mathtt{T}},\underline{\mathtt{S}},\{\}}(\mathcal{T})$, en $Imp^{\underline{\mathtt{T}},\underline{\mathtt{S}},\{\}}(\mathcal{M}^{\underline{\mathtt{T}}})$.*

Es claro que cada implementación canónica recoge en el soporte del género distinguido todos los registros formados por objetos funcionales que provienen de las implementaciones de $\mathcal{T}$, pero al mismo tiempo es lo único que recoge; por tanto, es la idea de mínimo objeto que aúna a todos los de una familia. La familia de i–transformaciones de la categoría $Imp^{\underline{\mathtt{T}},\underline{\mathtt{S}},\{\}}(\mathcal{M}^{\underline{\mathtt{T}}})$ que junto con la implementación canónica $\mathcal{I}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}$ forman el coproducto de las implementaciones de $\mathcal{T}$ con G-tipo $\underline{\mathtt{T}}$ y Ω-conjunto $\underline{\mathtt{S}}$ quedan definidas por las inclusiones entre los soportes asociados al género distinguido (única componente de las i–transformaciones que queda libre en la categoría $Imp^{\underline{\mathtt{T}},\underline{\mathtt{S}},\{\}}(\mathcal{M}^{\underline{\mathtt{T}}})$).

El resultado anterior justifica el hecho de que cada implementación canónica $\mathcal{I}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}$ contiene como datos a todas las tuplas de objetos funcionales que provienen de implementaciones de $\mathcal{T}$ con G-tipo $\underline{\mathtt{T}}$ y Ω-conjunto $\underline{\mathtt{S}}$.

Además, cada implementación canónica $\mathcal{I}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}$ posee otra propiedad universal en la categoría $Imp^{\underline{\mathtt{T}},\underline{\mathtt{S}},\{\}}(\mathcal{M}^{\underline{\mathtt{T}}})$ que recogemos en el siguiente teorema.

Teorema 3.7.9 *Existe al menos un morfismo de cualquier objeto de $Imp^{\underline{\mathtt{T}},\underline{\mathtt{S}},\{\}}(\mathcal{M}^{\underline{\mathtt{T}}})$ en $\mathcal{I}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}$.*

Demostración:

Sea $\mathcal{I}$ objeto de $Imp^{\underline{\mathtt{T}},\underline{\mathtt{S}},\{\}}(\mathcal{M}^{\underline{\mathtt{T}}})$, debemos definir una i–transformación $t^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}\colon \mathcal{I} \to \mathcal{I}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}$ de la categoría $Imp^{\underline{\mathtt{T}},\underline{\mathtt{S}},\{\}}(\mathcal{M}^{\underline{\mathtt{T}}})$. Por definición de los morfismos de esta categoría, es suficiente con dar una r–transformación $(t^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma}, id)$ de $\mathcal{R}^{\mathcal{I}}_{imp_\Sigma}$ en $\mathcal{R}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma}$. Sea $\mathcal{R}^{\mathcal{I}}_{imp_\Sigma}$=($\mathcal{D}^{\mathcal{I}}_{imp_\Sigma}$, $\mathcal{S}^{\mathcal{I}}_{imp_\Sigma}$, $=^{\mathcal{I}}_{imp_\Sigma}$, $\alpha^{\mathcal{I}}_{imp_\Sigma}$, $\mathcal{M}^{\underline{\mathtt{T}}}_{imp_\Sigma}$) la representación de $\mathcal{M}^{\underline{\mathtt{T}}}_{imp_\Sigma}$ en la implementación $\mathcal{I}$. Sea, para cada $\sigma \in \Omega$, $\mathtt{p}^{\mathcal{I}}_{imp\_\sigma} = (\mathtt{c}^{\mathcal{I}}_{imp\_\sigma}, \mathcal{S}^{\mathcal{I}}_{imp_\Sigma} \times \mathtt{S}^{\sigma}, \mathcal{D}^{\mathcal{I}}_{imp_\Sigma}, \mathtt{T}_{g_1}, \ldots, \mathtt{T}_{g_n}, \mathtt{T}_v)$ el programa que implementa la función $imp\_\sigma_{\mathcal{M}^{\underline{\mathtt{T}}}}$.

Definimos explícitamente $t^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma}$ como la función definida por el siguiente objeto funcional:

```
#'(lambda  (x)
     (make-IMP-g
        :imp-σ1   #' (lambda  (d1 ... dn1)
                              (funcall   c^I_imp_σ1  x  d1  ...  dn1))
        ⋮
        :imp-σm   #' (lambda  (d1 ... dnm)
                              (funcall   c^I_imp_σm  x  d1  ...  dnm))))
```

Debemos ver en primer lugar que $(t^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma}, id)$ es r–transformación. Para ello, lo primero que debemos asegurar es que, si $\mathtt{z} \in \mathcal{S}^{\mathcal{I}}_{imp_\Sigma}$, entonces $t^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma}(\mathtt{z}) \in \mathcal{S}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma}$. Como $\mathtt{z} \in \mathcal{S}^{\mathcal{I}}_{imp_\Sigma}$, se tiene que $\alpha^{\mathcal{I}}_{imp_\Sigma}(\mathtt{z}) \in \mathcal{M}^{\underline{\mathtt{T}},\underline{\mathtt{S}}}_{imp_\Sigma}$. Por tanto, existe al menos una implementación de $\mathcal{T}$ con G-tipo $\underline{\mathtt{T}}$, Ω-conjunto $\underline{\mathtt{S}}$ y tal que, las funciones definidas por sus programas son las de la tupla $\alpha^{\mathcal{I}}_{imp_\Sigma}(\mathtt{z})$. Denotamos por $\mathcal{J}^{\mathtt{z}}$ a una de esas implementaciones. Para asegurar que $t^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma}(\mathtt{z}) \in \mathcal{S}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma}$, debemos asegurar la existencia de una implementación de $\mathcal{T}$ cuyos programas tengan por códigos a las componentes de $t^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma}(\mathtt{z})$. Definimos, a partir de la implementación $\mathcal{J}^{\mathtt{z}}$, una implementación $\mathcal{I}^{\mathtt{z}}$ con las mismas representaciones, el mismo Ω-conjunto $\underline{\mathtt{S}}$ y siendo el código del programa que implementa a cada operación $\sigma\colon g_1 \ldots g_n \to g$ de Ω, el siguiente objeto funcional:

```
#'(lambda  (d1  ...  dn)
          (funcall   c^I_imp_σ  z  d1  ...  dn))
```

La principal propiedad que permite continuar con la demostración es la siguiente:

```
(funcall #'(lambda (x1 ... xn)  (funcall   c^I_imp_σ  z x1 ... xn)) d1  ... dn) ≡

≡ (funcall   c^I_imp_σ  z  d1  ...  dn)
```

La equivalencia anterior puede interpretarse como la β-reducción en λ-cálculo producida durante el proceso de evaluación, por lo que está bien definida y recogida en Common Lisp. (Una de las primeras referencias sobre λ-cálculo es [26], siendo referencias más recientes y en las que se puede encontrar lo relativo a la β-reducción [119] (página 34) y [7].) Esta propiedad es el punto clave de toda la demostración y, por tanto, del resultado recogido en el teorema.

Concretamente, una de las cuestiones que la propiedad anterior permite demostrar es que los programas de la implementación $\mathcal{I}^{\mathtt{z}}$ realmente lo son y que las representaciones de la implementación $\mathcal{J}^{\mathtt{z}}$ son coherentes con ellos. Además, los programas de ambas implementaciones, $\mathcal{J}^{\mathtt{z}}$ e $\mathcal{I}^{\mathtt{z}}$, son comportamentalmente iguales, es decir, definen las mismas funciones, propiedad obtenida por aplicación de la β-reducción. En esta situación, el resultado recogido en la proposición 3.5.7 nos permite asegurar que $\mathcal{I}^{\mathtt{z}}$ es una implementación de $\mathcal{T}$ con códigos de sus programas los objetos funcionales componentes de $t^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma}(\mathtt{z})$, y, por tanto, nos permite asegurar que $t^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma}(\mathtt{z})$ pertenece a $\mathcal{S}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma}$.

De lo anterior se deduce que los programas de las implementaciones $\mathcal{J}^{\mathtt{z}}$ y $\mathcal{I}^{\mathtt{z}}$ definen el mismo objeto de $\mathcal{M}^{\underline{\mathtt{T}},\underline{\mathtt{S}}}_{imp_\Sigma}$ (por ser iguales comportamentalmente) y se cumple la igualdad $\alpha^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma} \circ t^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma} = \alpha^{\mathcal{I}}_{imp_\Sigma}$, exigida en la definición de r–transformación. Por último, para asegurar que realmente $(t^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma}, id)$ es r–transformación es necesario verificar que preserva las igualdades de las representaciones, lo que se cumple ya que se ha tomado en $\mathcal{R}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma}$ la igualdad de la abstracción como igualdad de la representación, igualdad que es la comportamental y que, obviamente, es preservada.

Además, $t^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}$ es i–transformación. La descomposición de los dominios de definición de los programas como producto de una componente imp_Σ por un conjunto fijo $\mathtt{S}^\sigma$ para cada $\sigma \in \Omega$, junto con la relación entre los soportes de las representaciones $\left(t^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma}(\mathcal{S}^{\mathcal{I}}_{imp_\Sigma}) \subseteq \mathcal{S}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma}\right)$, permiten asegurar que todo elemento del dominio de definición de cualquier programa de la implementación $\mathcal{I}$, se transforma en un elemento del soporte del programa correspondiente en la implementación canónica. La igualdad $\alpha^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma} \circ t^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma} = \alpha^{\mathcal{I}}_{imp_\Sigma}$ asegura que los diagramas asociados a los operadores conmutan, es decir, que para cualquier $\sigma \in \Omega$ y para cualquier tupla $(\mathtt{x},\ \mathtt{d1},\ \ldots,\ \mathtt{dn}) \in \mathtt{S}^{\mathtt{P}^{\mathcal{I}}_{imp\_\sigma}}$, se verifica:

$$\mathtt{F}(\mathtt{p}^{\mathcal{I}}_{imp\_\sigma})(\mathtt{x},\ \mathtt{d1},\ \ldots,\ \mathtt{dn}) = \mathtt{F}(\mathtt{p}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp\_\sigma})(t^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma}(\mathtt{x}),\ \mathtt{d1},\ \ldots,\ \mathtt{dn})$$

lo que completa la demostración.

■

Recalcar que todo el peso de la demostración recae fundamentalmente en la propiedad señalada como la interpretación de la β-reducción del λ-cálculo durante el proceso de evaluación, ya que es la que permite obtener la igualdad $\alpha^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma} \circ t^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma} = \alpha^{\mathcal{I}}_{imp_\Sigma}$.

Esta última propiedad tiene la siguiente consecuencia:

Corolario 3.7.10 *Sean $\mathcal{I}$ y $\mathcal{J}$ implementaciones de $Imp^{\underline{\mathtt{T}},\underline{\mathtt{S}},\{\}}(\mathcal{M}^{\underline{\mathtt{T}}})$. Si la igualdad de la representación para el género imp_Σ en la implementación $\mathcal{J}$ es la igualdad de la abstracción (la comportamental), entonces como mucho existe una r–transformación de $\mathcal{I}$ en $\mathcal{J}$.*

Uniendo los dos últimos resultados obtenemos la siguiente caracterización de las implementaciones canónicas en la categoría $Imp^{\underline{\mathtt{T}},\underline{\mathtt{S}},\{\}}(\mathcal{M}^{\underline{\mathtt{T}}})$.

Teorema 3.7.11 *La implementación canónica* $\mathcal{I}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}$ *es objeto final en la categoría* $Imp^{\underline{\mathtt{T}},\underline{\mathtt{S}},\{\}}(\mathcal{M}^{\underline{\mathtt{T}}})$.

Si $Imp^{\underline{\mathtt{T}},\{\}}(\mathcal{M}^{\underline{\mathtt{T}}})$ denota a la categoría de implementaciones de $\mathcal{M}^{\underline{\mathtt{T}}}$ obtenida agrupando las categorías $Imp^{\underline{\mathtt{T}},\underline{\mathtt{S}},\{\}}(\mathcal{M}^{\underline{\mathtt{T}}})$, con $\underline{\mathtt{S}}$ variando en los Ω-conjuntos posibles, el resultado recogido en el último teorema permite obtener una caracterización de las implementaciones canónicas mediante la siguiente propiedad universal.

Corolario 3.7.12 *La familia de implementaciones canónicas* $\{\mathcal{I}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}\}_{\underline{\mathtt{S}} \text{ es } \Omega\text{-conjunto}}$ *es una familia final en la categoría* $Imp^{\underline{\mathtt{T}},\{\}}(\mathcal{M}^{\underline{\mathtt{T}}})$.

De forma análoga, si consideramos el TAD $\mathcal{T}_{imp}$ formado por todos los modelos $\mathcal{M}^{\underline{\mathtt{T}}}$, para cualquier G-conjunto $\underline{\mathtt{T}}$, y denotamos por $Imp^{\{\}}(\mathcal{T}_{imp})$ a la categoría de implementaciones obtenida por agrupación de las categorías $Imp^{\underline{\mathtt{T}},\{\}}(\mathcal{M}^{\underline{\mathtt{T}}})$, el siguiente resultado extiende el anterior.

Corolario 3.7.13 *La familia de implementaciones canónicas* $\{\mathcal{I}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}\}_{\underline{\mathtt{T}} \text{ es } G\text{-tipo},\underline{\mathtt{S}} \text{ es } \Omega\text{-conjunto}}$ *es una familia final en la categoría* $Imp^{\{\}}(\mathcal{T}_{imp})$.

Esta propiedad de finalidad de cada implementación canónica hace que sea ésta la más general entre todas las implementaciones de $\mathcal{M}^{\underline{\mathtt{T}}}$ con determinados tipos y soportes de los operadores, lo que desde el punto de vista de EAT se traduce en que una implementación canónica recoge a todas las implementaciones del TAD de partida con tipos y soportes de los operadores determinados y, permite trabajar con ellas como datos. El hecho de que la familia formada por todas las implementaciones canónicas sea final en la categoría $Imp^{\{\}}(\mathcal{T}_{imp})$, asegura que para cualquier implementación de $\mathcal{T}$ siempre existe una canónica que la posee como dato y que, además, podemos llegar a la familia canónica desde cualquier otra implementación de $\mathcal{M}^{\underline{\mathtt{T}}}$.

Las estructuras de datos utilizadas en los algoritmos de EAT corresponden básicamente a las ideas recogidas en las implementaciones canónicas, lo que justifica la afirmación de que la estrategia de implementación de EAT es apropiada y, en cierto sentido, general, para permitir la construcción de estructuras algebraicas, de cualquier naturaleza, en tiempo de ejecución.

Así, una consecuencia que obtenemos es que la forma de recoger el mayor número de representaciones de álgebras es por medio de una descripción funcional, consecuencia coherente con el análisis realizado en la parte algebraica.

Observando la demostración del teorema 3.7.9 vemos que no solo se demuestra la existencia de i–transformaciones (r–transformaciones) finales, sino que lo que se da, para cada implementación canónica, es un objeto funcional que define la correspondiente transformación. Esto motiva la siguiente definición.

Definición 3.7.14 *Una r–transformación* $(t, f) = (t_g, f_g)_{g \in G}$ *se dice* calculable *si para todo* $g \in G$, t_g *lo es.*

Es claro que, fijados un G-tipo $\underline{\mathtt{T}}$ y un Ω-conjunto $\underline{\mathtt{S}}$, la r–transformación final $t^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}$ de cada implementación $\mathcal{I}$ en la canónica $\mathcal{I}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}$ es calculable. Para cada $g \in G$, $id_{\mathtt{T}_g}$ es calculable, y el objeto funcional

$$\mathtt{c}^{\mathtt{T}_g} \equiv \mathtt{\#'(lambda\ \ (x)\ \ \ x)}$$

define un programa que lo demuestra: $(\mathtt{c}^{\mathtt{T}_g},\ \mathtt{T}_g,\ \mathtt{T}_g, \mathtt{T}_g)$.

Para el género imp_Σ, $t_{imp_\Sigma}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}$ también lo cumple. Existe un programa coherente con las representaciones literales, $\mathcal{R}^l_{\mathcal{D}^{\mathcal{I}}_{imp_\Sigma}}$ y $\mathcal{R}^l_{\mathcal{D}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma}}$, que implementa a la función $t_{imp_\Sigma}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}$, función que asocia a cada objeto `z` del soporte de la implementación $\mathcal{I}$, el registro formado por los códigos funcionales

```
#'(lambda  (d1  ...  dn)
      (funcall   c^I_imp_σ  z  d1  ...  dn))
```

Por ejemplo, un programa que la implementa es $\mathtt{p} = (\mathtt{c}^{t_{imp_\Sigma}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}}, \mathcal{S}^{\mathcal{I}}_{imp_\Sigma},\ \mathcal{D}^{\mathcal{I}}_{imp_\Sigma},\ \mathcal{D}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}_{imp_\Sigma})$, donde $\mathtt{c}^{t_{imp_\Sigma}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}}$ es el objeto funcional que define la r–transformación.

Por tanto, como todas las r–transformaciones finales son calculables, los resultados anteriores pueden extenderse a otras categorías. Denotamos por $Imp^{\underline{\mathtt{T}},\underline{\mathtt{S}},\{\}}_{calc}(\mathcal{M}^{\underline{\mathtt{T}}})$ a la subcategoría de $Imp^{\underline{\mathtt{T}},\underline{\mathtt{S}},\{\}}(\mathcal{M}^{\underline{\mathtt{T}}})$ con los mismos objetos y restringiendo las i–transformaciones (r–transformaciones) solo a las calculables. De manera análoga se definen las subcategorías $Imp^{\underline{\mathtt{T}},\{\}}_{calc}(\mathcal{M}^{\underline{\mathtt{T}}})$ e $Imp^{\underline{\mathtt{T}},\{\}}_{calc}(\mathcal{T}_{imp})$. Los resultados anteriores en estas categorías se leen de la siguiente manera.

Teorema 3.7.15 *La implementación canónica $\mathcal{I}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}$ es objeto final en la categoría $Imp^{\underline{\mathtt{T}},\underline{\mathtt{S}},\{\}}_{calc}(\mathcal{M}^{\underline{\mathtt{T}}})$.*

La familia de implementaciones canónicas $\{\mathcal{I}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}\}_{\underline{\mathtt{S}}\text{ es }\Omega\text{-conjunto}}$ es una familia final en la categoría $Imp^{\underline{\mathtt{T}},\{\}}_{calc}(\mathcal{M}^{\underline{\mathtt{T}}})$.

La familia de implementaciones canónicas $\{\mathcal{I}^{\underline{\mathtt{T}},\underline{\mathtt{S}},can}\}_{\underline{\mathtt{T}}\text{ es }G\text{-tipo},\underline{\mathtt{S}}\text{ es }\Omega\text{-conjunto}}$ es una familia final en la categoría $Imp^{\{\}}_{calc}(\mathcal{T}_{imp})$.

Observando la descripción de las implementaciones canónicas, es claro que la programación funcional juega un papel importante y de ahí la justificación de haber elegido un lenguaje de programación de naturaleza funcional como Common Lisp para la implementación de los algoritmos de EAT. Los resultados reposan no solo en las Matemáticas sino también en el lenguaje Common Lisp. Como ya comentamos al principio del capítulo es conocido que Common Lisp no posee una semántica formal, pero sí descripciones suficientemente precisas [116] que nos permiten apoyarnos en él. A lo largo del capítulo ya hemos ido comentando las características concretas de Common Lisp utilizadas. Concretamente, se usan propiedades de los registros en Common Lisp (`defstruct`) y, la "ecuación semántica" descrita en la demostración del teorema 3.7.9 como la interpretación de la β-reducción del λ-cálculo. Esta propiedad pertenece a la propia naturaleza funcional de Common Lisp y debe mantenerse en cualquier implementación correcta de Common Lisp.

En este punto dejamos una línea abierta en la que se busca generalizar los resultados recogidos en este capítulo para hacerlos independientes del lenguaje Common Lisp. Solamente apuntamos dos cuestiones. Por un lado, para modelar la noción de tipo concreto e incluir las funciones como datos y de forma independiente del lenguaje, será necesario trabajar en un contexto en el que puedan especificarse operadores funcionales de segundo orden [13]. Por otro lado,

para suplir la propiedad de Common Lisp que interpreta la β-reducción del λ-cálculo, parece clara la necesidad del lenguaje de disponer de una propiedad de tipo "cartesiano-cerrado".

En este capítulo hemos abordado el estudio de una de las características de las estructuras de datos de EAT: el uso de la codificación funcional. Hay una segunda característica de estas estructuras que no hemos abordado y que lo haremos en el siguiente capítulo: su naturaleza localmente efectiva, según la terminología debida a Sergeraert [102]. Esta segunda característica está relacionada con la capacidad de manipulación de estructuras de datos potencialmente infinitas.

3.8 Aplicaciones al Cálculo Simbólico

A lo largo del capítulo, hemos mostrado cómo, para cada TAD $\mathcal{T}$ y para cada G-tipo $\underline{\mathtt{T}}$, la Σ_{imp}-álgebra $\mathcal{M}^{\underline{\mathtt{T}}}$ recoge a todas las implementaciones de $\mathcal{T}$ con G-tipo $\underline{\mathtt{T}}$. Además, hemos probado que $\mathcal{M}^{\underline{\mathtt{T}}}$ es final en una categoría de álgebras que representan a familias de implementaciones del TAD de partida. Como ya se ha comentado, nuestro interés está en modelar la forma en la que en la práctica se pueden construir, en tiempo de ejecución, representaciones en la máquina de un TAD, lo que según hemos explicado conduce a la necesidad de encontrar adecuadas implementaciones de la Σ_{imp}-álgebra $\mathcal{M}^{\underline{\mathtt{T}}}$. En nuestro desarrollo teórico hemos llegado a dar una implementación "canónica" de $\mathcal{M}^{\underline{\mathtt{T}}}$, en el sentido de que ese modo de implementación da lugar a objetos finales en ciertas categorías de implementaciones. Lo anterior supone en particular que desde cualquier otra implementación existe un "paso natural" hasta ese modelo canónico.

A continuación, vamos a aplicar estos resultados teóricos generales sobre implementación a algunos de los tipos de datos que nos interesan por su importancia en el Cálculo Simbólico en Topología Algebraica. Trabajaremos sobre los mismos tipos de datos que desarrollamos en el capítulo anterior, aunque allí nuestro estudio se limitó a un nivel puramente algebraico. Así, construiremos implementaciones canónicas de los conjuntos simpliciales y de los complejos de cadenas.

3.8.1 Conjuntos simpliciales

Al final del capítulo anterior estudiamos lo que según nuestro criterio es un buen formalismo para trabajar con conjuntos simpliciales, el TAD $\mathcal{T}_{\mathtt{CS}} = \langle \mathtt{CS}, \mathcal{C}(\mathtt{CS}) \rangle$ donde la signatura $\mathtt{CS}$ se define por

$$\delta \quad : \quad nat \quad nat \quad smp \quad \rightarrow \quad smp$$

$$\eta \quad : \quad nat \quad nat \quad smp \quad \rightarrow \quad smp$$

y $\mathcal{C}(\mathtt{CS})$ es una categoría que resulta equivalente a la categoría de los conjuntos simpliciales (página 105). (Un estudio menos detallado de este ejemplo, tanto en el nivel algebraico como en el de implementación, fue publicado en [66].)

La familia de tipos $\underline{\mathtt{T}}$ que vamos a fijar aquí refleja fielmente los tipos de datos que se usan en EAT para implementar a los conjuntos simpliciales. Recordando lo que nos interesa del estudio de la parte algebraica de este ejemplo, vimos que una forma adecuada de trabajar con conjuntos simplicicales, es usar la representación de los símplices como abstractos, de forma que

cada símplice puede expresarse como una sucesión de degeneraciones de un símplice geométrico. Esto se traduce en que si $K = \{K^n\}_{n\in\mathbb{N}}$ es un conjunto graduado que en cada dimensión contiene al conjunto de símplices geométricos, podemos considerar el conjunto I^K_{absmp}, conjunto que recoge la representación de cada símplice como abstracto, y cuya descripción es la siguiente

$$I^K_{absmp} = \Big\{\big\langle (j_k,\ldots,j_1), a\big\rangle \;\mid\; \exists\ n \in \mathbb{N} \text{ tal que } a \in K^n,\ k \in \mathbb{N}, j_i \in \mathbb{N} \text{ para todo } i = 1,\ldots,k, \text{ y } 0 \le j_1 < \cdots < j_k \le n+k-1\Big\}$$

Es importante, desde un punto de vista práctico, señalar que este conjunto nos proporciona un patrón sintáctico, en el que nos basaremos para elegir un tipo Lisp para el género *smp*, con el que se representarán los símplices en la máquina. Así, pensaremos en un símplice como un par formado por una lista estrictamente decreciente o vacía de enteros (que representa los operadores de degeneración aplicados), y por otro objeto, que representa al símplice geométrico. Con esta idea consideramos en primer lugar el siguiente tipo:

```
(defstruct  asm  dop  gsm)
```

Tomamos como $\mathtt{T}_{smp}$ aquellos datos de tipo `asm` tales que el campo `dop` es una lista de datos de tipo `integer` estrictamente decreciente o vacía, y `gsm` es cualquier objeto Lisp. Para el género *nat*, como es natural, fijamos el tipo `(satisfies naturalp)`, de forma que el paso de las Matemáticas a la máquina suponen, en este caso, un mero cambio del marco de referencia.

Así, la $\mathtt{CS}_{imp}$-álgebra $\mathcal{M}^{\underline{\mathtt{T}}}$, final en la categoría de $\mathtt{CS}_{imp}$-álgebras definidas a partir de implementaciones de conjuntos simpliciales del TAD $\mathcal{T}_{\mathtt{CS}}$ con tipos `(satisfies naturalp)` y $\mathtt{T}_{smp}$, y en la que se han restringido los morfismos (su existencia la asegura el teorema 3.7.2), tiene como soportes para los géneros *nat* y *smp* los tipos fijados, y como soporte asociado al género $imp_{\mathtt{CS}}$, el siguiente conjunto

$$\mathcal{M}^{\underline{\mathtt{T}}}_{imp_{\mathtt{CS}}} = \Big\{(\mathtt{f}_\delta, \mathtt{f}_\eta) \;\mid\; \exists\ \mathcal{I} = \big(\underline{\mathcal{R}}, \mathtt{p}^{\mathcal{I}}_\delta, \mathtt{p}^{\mathcal{I}}_\eta\big) \in Obj(Imp^{\underline{\mathtt{T}}}(\mathcal{T}_{\mathtt{CS}})) \text{ tal que } \mathtt{f}_\delta = F(\mathtt{p}^{\mathcal{I}}_\delta) \text{ y } \mathtt{f}_\eta = F(\mathtt{p}^{\mathcal{I}}_\eta)\Big\}$$

donde $Imp^{\underline{\mathtt{T}}}(\mathcal{T}_{\mathtt{CS}})$ denota a la categoría de implementaciones de $\mathcal{T}_{\mathtt{CS}}$ con $G_{\mathtt{CS}}$-tipo $\underline{\mathtt{T}}$. Recordar que las operaciones del álgebra anterior vienen dadas por las proyecciones en las componentes adecuadas.

Hemos visto que hay una implementación canónica de la $\mathtt{CS}_{imp}$-álgebra $\mathcal{M}^{\underline{\mathtt{T}}}$ por cada posible elección para los soportes de los programas de las implementaciones de los conjuntos simpliciales. En nuestro caso, los dominios de definición elegidos para las álgebras que constituyen el TAD $\mathcal{T}_{\mathtt{CS}}$ siguen todos un mismo patrón, que vamos a tener en cuenta para fijar los soportes que nos interesan en la práctica. Para ello, fijado un conjunto graduado $K = \{K^n\}_{n\in\mathbb{N}}$ en $\mathcal{U}$ solo vamos a considerar implementaciones del TAD $\mathcal{T}_{\mathtt{CS}}$ que tengan como soportes de los programas los siguientes conjuntos

$\cdot\ \mathtt{S}^{K,\delta} = \Big\{(\mathtt{i},\mathtt{n},\mathtt{x}) \in$ `(satisfies naturalp)` $\times$ `(satisfies naturalp)` $\times \mathtt{T}_{smp} \mid$ `(<= i n)` $\wedge$ `(> n 0)` $\wedge\ \exists$`j` $\in$ `(satisfies naturalp)` tal que `(asm-gsm x)` $\in K^{\mathtt{j}} \wedge$ `(= (+ (length (asm-dop x)) j) n)` $\Big\}$

· $\mathtt{S}^{K,\eta} = \Big\{(\mathtt{i},\mathtt{n},\mathtt{x}) \in \mathtt{(satisfies\ naturalp)} \times \mathtt{(satisfies\ naturalp)} \times \mathtt{T}_{smp} \mid$
$\mathtt{(<=\ i\ n)} \wedge\ \exists \mathtt{j} \in \mathtt{(satisfies\ naturalp)}$ tal que $\mathtt{(asm\text{-}gsm\ x)} \in K^{\mathtt{j}}$
$\wedge\ \mathtt{(=\ (+\ (length\ (asm\text{-}dop\ x))\ j)\ n)}\ \Big\}$

Los soportes anteriores se obtienen mediante una mera traslación de los dominios de los operadores de las álgebras de $\mathcal{T}_{\mathtt{CS}}$. Así, podemos pensar que las definiciones anteriores solo dependen del conjunto graduado K. Por ello, a diferencia de lo que hemos venido haciendo en la parte teórica del capítulo, no va a aparecer explícitamente el $\Omega_{\mathtt{CS}}$-conjunto fijado sino solamente el conjunto graduado K. Denotamos por $\mathcal{I}^{\underline{\mathtt{T}},K,can}$ a la implementación canónica de $\mathcal{M}^{\underline{\mathtt{T}}}$, que, en este caso, viene definida como sigue

- Para el género *nat* se toma la representación literal sobre el tipo `(satisfies naturalp)`; para el género *smp* la literal sobre $\mathtt{T}_{smp}$.
- Para definir la representación del género $imp_{\mathtt{CS}}$, $\mathcal{R}^{\underline{\mathtt{T}},K,can}_{imp_{\mathtt{CS}}}$, consideramos el tipo

$$\mathtt{(defstruct\ \ IMP\text{-}CS\ \ imp\text{-}}\delta\ \mathtt{\ imp\text{-}}\eta\mathtt{)}$$

y tomamos como dominio aquellos datos de la estructura anterior que satisfagan el predicado `D-can-CS-p` que decide si los dos campos de un dato de tipo `IMP-CS` son o no objetos funcionales. Este predicado viene dado del siguiente modo

```
(defun D-can-CS-p (x)
     (and (typep x 'IMP-CS)
          (functionp (IMP-CS-imp-δ x))
          (functionp (IMP-CS-imp-η x))))
```

Como soporte de la representación se toma el siguiente subconjunto del dominio anterior:

$$\mathcal{S}^{\underline{\mathtt{T}},K,can}_{imp_{\mathtt{CS}}} = \Big\{\mathtt{x} \in \mathtt{(satisfies\ D\text{-}can\text{-}CS\text{-}p)} \mid\ \exists\ \mathcal{I} = \big(\underline{\mathcal{R}}, \mathtt{p}^{\mathcal{I}}_{\delta}, \mathtt{p}^{\mathcal{I}}_{\eta}\big) \in Obj(Imp^{\underline{\mathtt{T}}}(\mathcal{T}_{\mathtt{CS}}))$$
$$\text{con } \Omega_{\mathtt{CS}}\text{-conjunto } (\mathtt{S}^{K,\delta}, \mathtt{S}^{K,\eta}) \text{ y tal que } \mathtt{(eq\ \ c}^{\mathtt{p}^{\mathcal{I}}_{\delta}}\ \mathtt{\ (IMP\text{-}CS\text{-}imp\text{-}}\delta\ \mathtt{x))} \text{ y}$$
$$\mathtt{(eq\ \ c}^{\mathtt{p}^{\mathcal{I}}_{\eta}}\ \mathtt{\ (IMP\text{-}CS\text{-}imp\text{-}}\eta\ \mathtt{x))}\ \Big\}$$

La función de abstracción $\alpha^{\underline{\mathtt{T}},K,can}_{imp_{\mathtt{CS}}} \colon \mathcal{S}^{\underline{\mathtt{T}},K,can}_{imp_{\mathtt{CS}}} \to \mathcal{M}^{\underline{\mathtt{T}}}_{imp_{\mathtt{CS}}}$ lleva cada registro de códigos funcionales al par de funciones, con dominios $\mathtt{S}^{K,\delta}$ y $\mathtt{S}^{K,\eta}$, definidas por los códigos. Es decir, para cada $\mathtt{x} \in \mathcal{S}^{\underline{\mathtt{T}},K,can}_{imp_{\mathtt{CS}}}$, $\alpha^{\underline{\mathtt{T}},K,can}_{imp_{\mathtt{CS}}}(\mathtt{x}) := (\mathtt{f}^{\mathtt{x}}_{\delta}, \mathtt{f}^{\mathtt{x}}_{\eta})$, siendo

· $\mathtt{f}^{\mathtt{x}}_{\delta}$: $\mathtt{(satisfies\ naturalp)} \times \mathtt{(satisfies\ naturalp)} \times \mathtt{T}_{smp} \to \mathtt{T}_{smp}$ la función con dominio $\mathtt{S}^{K,\delta}$ y definida por

$$\mathtt{f}^{\mathtt{x}}_{\delta}(\mathtt{i},\mathtt{n},\mathtt{z}) := \mathtt{(funcall\ \ (IMP\text{-}CS\text{-}imp\text{-}}\delta\ \mathtt{x)\ \ i\ \ n\ \ z)}$$

· Análogamente se define $\mathtt{f}^{\mathtt{x}}_{\eta}$.

La función de abstracción está bien definida ya que la implementación de $\mathcal{T}_{\mathtt{CS}}$ que hace que $\mathtt{x}$ esté en el soporte $\mathcal{S}_{imp_{\mathtt{CS}}}^{\underline{\mathtt{T}},K,can}$, hace que $(\mathtt{f}_\delta^{\mathtt{x}}, \mathtt{f}_\eta^{\mathtt{x}})$ esté en $\mathcal{M}_{imp_{\mathtt{CS}}}^{\underline{\mathtt{T}}}$. Como igualdad de representación tomamos la de la abstracción.

- Como programas que implementan a las operaciones, consideramos: $\mathtt{p}_{imp\_\sigma}^{\underline{\mathtt{T}},K,can}$=($\mathtt{c}^{\mathtt{p}_{imp\_\sigma}^{\underline{\mathtt{T}},K,can}}$, $\mathcal{S}_{imp_{\mathtt{CS}}}^{\underline{\mathtt{T}},K,can} \times \mathtt{S}^{K,\sigma}$, $\mathcal{D}_{imp_{\mathtt{CS}}}^{\underline{\mathtt{T}},K,can}$, `(satisfies naturalp)`, `(satisfies naturalp)`, $\mathtt{T}_{smp}$, $\mathtt{T}_{smp}$), siendo el código asociado el siguiente objeto funcional

$$\mathtt{c}^{\mathtt{p}_{imp\_\sigma}^{\underline{\mathtt{T}},K,can}} \equiv \texttt{\#'(lambda (z i n x)}$$
$$\texttt{(funcall (IMP-CS-imp-}\sigma\ \texttt{z) i n x))}$$

con $\sigma \in \{\delta, \eta\}$.

El resultado recogido en 3.7.11, asegura que la implementación $\mathcal{I}^{\underline{\mathtt{T}},K,can}$ es objeto final en una subcategoría adecuada de la categoría de implementaciones de $\mathcal{M}^{\underline{\mathtt{T}}}$ (resultado que ya incluimos en [66]).

Además, en el ejemplo 3.7.5, dimos otra implementación $\mathcal{I}^{K\cup}$ de la $\mathtt{CS}_{imp}$-álgebra $\mathcal{M}^{\underline{\mathtt{T}}}$ (página 170). La i–transformación final entre ambas implementaciones viene dada, para cada $\mathtt{m} \in$ `(satisfies naturalp)` $\setminus\{0\}$ (recordar que el soporte del género $imp_{\mathtt{CS}}$ en $\mathcal{I}^{K\cup}$ era el tipo de los naturales Lisp) por

```
#'(lambda  (m)
     (make-IMP-CS
        :imp-δ   #' (lambda (m)
                      (funcall   c_{imp_δ}^{K∪}  m  i  n  x))
        :imp-η   #' (lambda (m)
                      (funcall   c_{imp_η}^{K∪}  m  i  n  x))))
```

Para la implementación $\widetilde{\mathcal{I}}^{K\cup}$ de la misma $\mathtt{CS}_{imp}$-álgebra (página 171), la propia i–transformación anterior es también el morfismo final. Así, las implementaciones $\mathcal{I}^{K\cup}$ e $\widetilde{\mathcal{I}}^{K\cup}$ dan lugar a la misma tupla de códigos funcionales. Esto demuestra que existe sobrecarga de los objetos funcionales que aparecen en el objeto final. Por ejemplo, en este caso, un mismo par de códigos representa a una esfera en cualquier dimensión, y además, codificada de dos formas diferentes. Lo anterior pone de manifiesto que la representación en la máquina de cualquier objeto, ha de ir siempre acompañada, como mínimo, de una función de abstracción que está en la mente del programador, y que le permite interpretar los resultados de los cálculos realizados. Como se desprende de la discusión anterior, la implementación canónica se puede interpretar como el objeto más general de toda la categoría de implementaciones, en el sentido de que, desde cualquier otra implementación, va a existir siempre un paso natural hasta ella (mediante una i–transformación calculable).

Hasta aquí hemos aplicado estrictamente la teoría dada en el capítulo. Sin embargo, el estar en un ejemplo concreto disponemos de información adicional, lo que nos va a permitir avanzar un poco más de lo visto para el caso general.

El conjunto I_{absmp}^{K} nos proporciona un patrón para trabajar con la representación de los

símplices como abstractos, siempre que se disponga de un conjunto graduado K como base para los símplices geométricos. Ahora bien, si no fijamos ningún conjunto graduado, un patrón general para los símplices abstractos lo proporciona el conjunto

$$SmAb = \Big\{ \big\langle (j_k, \ldots, j_1), a \big\rangle \;\mid\; k \in \mathbb{N}, j_i \in \mathbb{N}\ \forall i = 1, \ldots, k,\ 0 \leq j_1 < \cdots < j_k \Big\}$$

donde a es cualquier objeto del universo algebraico que se considere. Así, I^K_{absmp} es una particularización concreta de $SmAb$ para un conjunto de símplices geométricos fijo. Como queremos permanecer próximos a EAT, y en él se trabaja con símplices abstractos, en la práctica no nos van a interesar todas las álgebras de la categoría $\mathcal{C}(\mathtt{CS})$, sino solo aquellas cuyo conjunto soporte para el género *smp* pueda verse como un subconjunto de $SmAb$. Así, denotamos por $\mathcal{C}^{SmAb}(\mathtt{CS})$ a la subcategoría de $\mathcal{C}(\mathtt{CS})$ con objetos las álgebras con conjunto soporte para el género *smp* un subconjunto de $SmAb$. Extendiendo la notación anterior, $\mathcal{T}^{SmAb}_{\mathtt{CS}}$ denotará al TAD dado por $\langle \mathtt{CS}, \mathcal{C}^{SmAb}(\mathtt{CS}) \rangle$.

Respecto del $G_{\mathtt{CS}}$-tipo, fijamos de nuevo `(satisfies naturalp)` y $\mathtt{T}_{smp}$ como tipos para *nat* y *smp*, respectivamente.

Paralelamente, no nos va a interesar considerar la totalidad de la categoría de implementaciones. Todas las álgebras de $\mathcal{C}^{SmAb}(\mathtt{CS})$ tienen como soportes $\mathbb{N}$ y un subconjunto de $SmAb$, y los tipos fijados para las implementaciones son `(satisfies naturalp)` y $\mathtt{T}_{smp}$. Es claro que las elecciones anteriores provienen de pensar en una función de abstracción $\alpha_{smp} \colon \mathtt{T}_{smp} \to SmAb$ que sea un mero cambio del marco de referencia (paso de Common Lisp a las Matemáticas). Así, definimos $Imp^{\underline{\mathtt{T}}^{nat}}(\mathcal{T}^{SmAb}_{\mathtt{CS}})$ como la subcategoría plena de $Imp(\mathcal{T}^{SmAb}_{\mathtt{CS}})$ con objetos las implementaciones con tipos `(satisfies naturalp)` y $\mathtt{T}_{smp}$ y representaciones naturales sobre los tipos anteriores. Partiendo de esta categoría de implementaciones, consideramos la correspondiente $\mathtt{CS}_{imp}$-álgebra $\mathcal{M}^{\underline{\mathtt{T}}^{nat}, SmAb}$ que obviamente tiene por soportes para *nat* y *smp*, `(satisfies naturalp)` y $\mathtt{T}_{smp}$, respectivamente, y para el género $imp_{\mathtt{CS}}$ el siguiente conjunto

$$\mathcal{M}^{\underline{\mathtt{T}}^{nat}, SmAb}_{imp_{\mathtt{CS}}} = \Big\{ (\mathtt{f}_\delta, \mathtt{f}_\eta) \;\mid\; \exists\ \mathcal{I} = \big(\underline{\mathcal{R}}, \mathtt{p}^{\mathcal{I}}_\delta, \mathtt{p}^{\mathcal{I}}_\eta\big) \in Obj(Imp^{\underline{\mathtt{T}}^{nat}}(\mathcal{T}^{SmAb}_{\mathtt{CS}})) \ \text{ tal que } \mathtt{f}_\delta = F(\mathtt{p}^{\mathcal{I}}_\delta) \text{ y } \mathtt{f}_\eta = F(\mathtt{p}^{\mathcal{I}}_\eta) \Big\}$$

Como siempre, las operaciones serán las proyecciones en las componentes correspondientes. Al igual que hemos hecho en el caso general, fijando un conjunto graduado $K = \{K^n\}_{n \in \mathbb{N}}$ en $\mathcal{U}$, consideraremos solo implementaciones de $Imp^{\underline{\mathtt{T}}^{nat}}(\mathcal{T}^{SmAb}_{\mathtt{CS}})$ que tengan como soportes de los programas a los correspondientes conjuntos $\mathtt{S}^{K,\delta}$ y $\mathtt{S}^{K,\eta}$. En particular, la elección de las representaciones naturales lleva consigo que el soporte de la representación para el género *nat* sea todo `(satisfies naturalp)` y el soporte de la representación para el género *smp* sea el conjunto $\big\{ \mathtt{x} \in \mathtt{T}_{smp} \;\mid\; \exists \mathtt{j} \in$ `(satisfies naturalp)` con `(asm-gsm x)` $\in K^{\mathtt{j}} \big\}$. (Observar que la elección de los conjuntos $\mathtt{S}^{K,\delta}$ y $\mathtt{S}^{K,\eta}$ como soportes de los programas es natural, ya que son el paso al universo Common Lisp de los dominios de definición de las operaciones de las álgebras de $\mathcal{C}^{SmAb}(\mathtt{CS})$.) En esta situación, podemos definir la correspondiente implementación canónica $\mathcal{I}^{\underline{\mathtt{T}}^{nat}, K, SmAb, can}$ del álgebra final $\mathcal{M}^{\underline{\mathtt{T}}^{nat}, SmAb}$.

En la parte algebraica se ha visto que cada álgebra de $\mathcal{C}^{SmAb}(\mathtt{CS})$ tiene determinado el operador de degeneración, que, además, viene dado por la función que hemos denominado η_{I^K}, para cada conjunto graduado K. Recordar que esta función $\eta_{I^K} \colon \mathbb{N} \times \mathbb{N} \times I^K_{absmp} \to I^K_{absmp}$ tiene por dominio $\big\{ \big(i, n, \langle (j_k, \ldots, j_1), a \rangle\big) \;\mid\; 0 \leq i \leq n, \quad \exists\, l \in \mathbb{N}$ tal que $a \in K^l$ siendo $l + k = n \big\}$,

y se define del siguiente modo

$$\eta_{I^K}\big(i, n, \langle (j_k, \ldots, j_1), a\rangle\big) := \langle (j_k + 1, \ldots, j_l + 1, i, j_{l-1}, \ldots, j_1), a\rangle$$

siendo l tal que $j_{l-1} < i \leq j_l$.

(Como ya comentamos, esta definición equivale a añadir i al final de la lista de degeneraciones, y aplicar la condición que verifican los conjuntos simpliciales y que permite intercambiar el orden de dos operadores de degeneración, hasta obtener una lista estrictamente decreciente.)

Lo anterior, unido a que hemos impuesto que las funciones de abstracción de las implementaciones consideradas sean las naturales, hace que todas las funciones definidas por los programas que implementan a los operadores de degeneración de las álgebras de $\mathcal{C}^{SmAb}(\mathtt{CS})$ (definidas sobre el mismo conjunto graduado) deben ser la misma, la dada por η^K: (`satisfies naturalp`) × (`satisfies naturalp`) × $\mathtt{T}_{smp} \to \mathtt{T}_{smp}$ con comportamiento análogo al de la función η_{I^K}. Así, el dominio de esta función es:

$$Def(\eta^K) = \Big\{(\mathtt{i},\mathtt{n},\mathtt{x}) \mid \texttt{(<= i n)} \wedge \exists\ \mathtt{j} \in \texttt{(satisfies naturalp)} \text{ tal que } \texttt{(asm-gsm x)} \in K^j \text{ y } \texttt{(= (+ (length (asm-dop x)) j) n)}\Big\}$$

y su comportamiento viene dado por $\eta^K(\mathtt{i},\mathtt{n},\mathtt{x}) := \mathtt{y}$ tal que si `(asm-dop x)` es `equal` a $(\mathtt{j_k} \ldots \mathtt{j_1})$ entonces `(asm-dop y)` es `equal` a $(\mathtt{j_k + 1} \ldots \mathtt{j_1 + 1\ i\ j_{l-1}} \ldots \mathtt{j_1})$, siendo además, `(asm-dop x) eq` a `(asm-dop y)`. Las igualdades que se utilizan en la definición anterior son las específicas de cada tipo, concretamente, `equal` es la natural para comparar listas y `eq` para comparar cualquier objeto Lisp.

Es claro que la función anterior es calculable y, por tanto, podemos construir un objeto funcional que nos permita definir un programa, que denotamos por $\mathtt{p}^K_\eta$, con dominio el de la función anterior y cuyo comportamiento sobre este dominio venga descrito por η^K (la función viene de trasladar η_{I^K} a los tipos fijados para representar, de forma natural, sus dominios).

Si observamos la definición de cada función η^K nos damos cuenta que es posible definir una función η^{SmAb} que, particularizada a cada conjunto graduado K, se comporte como η^K

$$\eta^{SmAb}\colon (\texttt{satisfies naturalp}) \times (\texttt{satisfies naturalp}) \times \mathtt{T}_{smp} \to \mathtt{T}_{smp}$$

con dominio $Def(\eta^{SmAb}) = \big\{(\mathtt{i},\mathtt{n},\mathtt{x}) \mid \texttt{(<= i n)}\big\}$ y comportamiento $\eta^{SmAb}(\mathtt{i},\mathtt{n},\mathtt{x}) := \mathtt{y}$ tal que si `(asm-dop x)` es `equal` a $(\mathtt{j_k} \ldots \mathtt{j_1})$ entonces `(asm-dop y)` es `equal` a $(\mathtt{j_k + 1} \ldots \mathtt{j_1 + 1\ i\ j_{l-1}} \ldots \mathtt{j_1})$, siendo además, `(asm-gsm x) eq` a `(asm-gsm y)`.

La función η^{SmAb} es calculable y, por tanto, podemos afirmar que existe un modo universal de programar los operadores de degeneración, que incluso es independiente del conjunto simplicial implementado. Así, consideramos el programa $\mathtt{p}_{\eta^{SmAb}}$=($\mathtt{c}^{\mathtt{p}_{\eta^{SmAb}}}$, $Def(\eta^{SmAb})$, `(satisfies naturalp)`, `(satisfies naturalp)`, $\mathtt{T}_{smp}$, $\mathtt{T}_{smp}$), siendo $\mathtt{c}^{\mathtt{p}_{\eta^{SmAb}}}$ cualquier objeto funcional que sobre $Def(\eta^{SmAb})$ defina la función η^{SmAb}.

Desde el punto de vista algebraico, lo anterior se corresponde con el hecho de que se puede dar una descripción del objeto final, según vimos en el capítulo anterior, eliminando de las tuplas de funciones las componentes procedentes de operadores que puedan considerarse visibles para todas las $\mathtt{CS}_{imp}$-álgebras que se estén considerando, porque no dependen de la componente del género distinguido.

Aplicando lo anterior a implementaciones, podemos dar otra descripción funcional del objeto que recoge a las implementaciones de la categoría $Imp^{\mathtt{T}^{nat}}(\mathcal{T}_{\mathtt{CS}}^{SmAb})$, más reducida que la dada para $\mathcal{M}^{\mathtt{T}^{nat},SmAb}$. Así, denotamos por $\mathcal{M}^{\mathtt{T}^{nat},SmAb,red}$ a la $\mathtt{CS}_{imp}$-álgebra con soportes `(satisfies naturalp)` y $\mathtt{T}_{smp}$ para los géneros nat y smp, y para el género $imp_{\mathtt{CS}}$, el siguiente conjunto

$$\mathcal{M}_{imp_{\mathtt{CS}}}^{\mathtt{T}^{nat},SmAb,red} = \left\{ \mathtt{f}_\delta \;\mid\; \exists\, \mathcal{I} = \left(\underline{\mathcal{R}}, \mathtt{p}_\delta^{\mathcal{I}}, \mathtt{p}_\eta^{\mathcal{I}}\right) \in Obj(Imp^{\mathtt{T}^{nat}}(\mathcal{T}_{\mathtt{CS}}^{SmAb})) \text{ tal que } \mathtt{f}_\delta = F(\mathtt{p}_\delta^{\mathcal{I}}) \right\}$$

La operación $imp\_\delta_{\mathcal{M}^{\mathtt{T}^{nat},SmAb,red}}$ se define como la aplicación de la función que es el primer argumento al resto, mientras que la operación $imp\_\eta_{\mathcal{M}^{\mathtt{T}^{nat},SmAb,red}}$ viene dada por la función η^{SmAb}.

Las $\mathtt{CS}_{imp}$-álgebras $\mathcal{M}^{\mathtt{T}^{nat},SmAb}$ y $\mathcal{M}^{\mathtt{T}^{nat},SmAb,red}$ son isomorfas en la categoría de $\mathtt{CS}_{imp}$-álgebras definida a partir de las implementaciones de $Imp^{\mathtt{T}^{nat}}(\mathcal{T}_{\mathtt{CS}}^{SmAb})$.

Vamos a dar una implementación canónica de $\mathcal{M}^{\mathtt{T}^{nat},SmAb,red}$. Comenzamos por fijar los soportes de los programas de las implementaciones de $Imp^{\mathtt{T}^{nat}}(\mathcal{T}_{\mathtt{CS}}^{SmAb})$ que consideraremos. Así, sea K un conjunto graduado en $\mathcal{U}$ y consideramos los conjuntos $\mathtt{S}^{K,\delta}$ y $\mathtt{S}^{K,\eta}$ (definidos en la página 184). Denotamos por $\mathcal{I}^{\mathtt{T}^{nat},K,SmAb,red,can}$ a la implementación de $\mathcal{M}^{\mathtt{T}^{nat},SmAb,red}$ (implementación que ya estudiamos en [66], aunque con menos detalle) y que viene dada como sigue:

- Representaciones literales para los tipos `(satisfies naturalp)` y $\mathtt{T}_{smp}$.
- Para el género $imp_{\mathtt{CS}}$, tomamos como dominio el tipo Lisp `function`. Como soporte consideramos el siguiente subconjunto del dominio anterior:

$$\mathcal{S}_{imp_{\mathtt{CS}}}^{\mathtt{T}^{nat},K,SmAb,red,can} = \Big\{ \mathtt{x} \in \mathtt{function} \;\mid\; \exists\, \mathcal{I} = \left(\underline{\mathcal{R}}, \mathtt{p}_\delta^{\mathcal{I}}, \mathtt{p}_\eta^{\mathcal{I}}\right) \in Obj(Imp^{\mathtt{T}^{nat}}(\mathcal{T}_{\mathtt{CS}}^{SmAb})) \text{ con } \Omega_{\mathtt{CS}}\text{-conjunto } (\mathtt{S}^{K,\delta}, \mathtt{S}^{K,\eta}) \text{ y tal que } \mathtt{(eq\ x\ c}^{\mathtt{p}_\delta^{\mathcal{I}}}\mathtt{)} \Big\}$$

  La función de abstracción $\alpha_{imp_{\mathtt{CS}}}^{\mathtt{T}^{nat},K,SmAb,red,can} : \mathcal{S}_{imp_{\mathtt{CS}}}^{\mathtt{T}^{nat},K,SmAb,red,can} \to \mathcal{M}_{imp_{\mathtt{CS}}}^{\mathtt{T}^{nat},SmAb,red}$ lleva cada objeto funcional a la función que sobre $\mathtt{S}^{K,\delta}$ define ese objeto. Como igualdad de representación tomamos la de la abstracción.

- $\mathtt{p}_{imp\_\delta}^{\mathtt{T}^{nat},K,SmAb,red,can} = (\mathtt{c}^{\mathtt{p}_{imp\_\delta}^{\mathtt{T}^{nat},K,SmAb,red,can}}, \mathcal{S}_{imp_{\mathtt{CS}}}^{\mathtt{T}^{nat},K,SmAb,red,can} \times \mathtt{S}^{K,\delta}, \mathcal{D}_{imp_{\mathtt{CS}}}^{\mathtt{T}^{nat},K,SmAb,red,can}, \mathtt{(satisfies\ naturalp)}, \mathtt{(satisfies\ naturalp)}, \mathtt{T}_{smp}, \mathtt{T}_{smp})$, siendo el código asociado el siguiente objeto funcional

$$\mathtt{c}^{\mathtt{p}_{imp\_\delta}^{\mathtt{T}^{nat},K,SmAb,red,can}} \equiv \mathtt{\#'(lambda\ (z\ i\ n\ x)\ (funcall\ z\ i\ n\ x))}$$

- $\mathtt{p}_{imp\_\eta}^{\mathtt{T}^{nat},K,SmAb,red,can} = (\mathtt{c}^{\mathtt{p}_{imp\_\eta}^{\mathtt{T}^{nat},K,SmAb,red,can}}, \mathcal{S}_{imp_{\mathtt{CS}}}^{\mathtt{T}^{nat},K,SmAb,red,can} \times \mathtt{S}^{K,\eta}, \mathcal{D}_{imp_{\mathtt{CS}}}^{\mathtt{T}^{nat},K,SmAb,red,can}, \mathtt{(satisfies\ naturalp)}, \mathtt{(satisfies\ naturalp)}, \mathtt{T}_{smp}, \mathtt{T}_{smp})$, con el siguiente objeto funcional como código

$$\mathtt{c}^{\mathtt{p}_{imp\_\eta}^{\mathtt{T}^{nat},K,SmAb,red,can}} \equiv \mathtt{\#'(lambda\ (z\ i\ n\ x)\ (funcall\ c}^{\mathtt{p}_{\eta^{SmAb}}}\ \mathtt{i\ n\ x))}$$

El isomorfismo entre $\mathcal{M}^{\mathtt{T}^{nat},SmAb}$ y $\mathcal{M}^{\mathtt{T}^{nat},SmAb,red}$, induce i–transformaciones entre las implementaciones canónicas asociadas a cada una de ellas. En la parte de los modelos esas i–transformaciones vienen dadas por el isomorfismo entre las álgebras y en la parte de los tipos por las identidades sobre `(satisfies naturalp)` y $\mathtt{T}_{smp}$. Queda por dar las transformaciones entre los conjuntos soporte del género $imp_{\mathtt{CS}}$ que definimos como sigue

- $t_{imp_{\mathtt{CS}}}\colon \mathcal{D}^{\mathtt{T}^{nat},K,SmAb,can}_{imp_{\mathtt{CS}}} \to \mathcal{D}^{\mathtt{T}^{nat},K,SmAb,red,can}_{imp_{\mathtt{CS}}}$ definida para cada $\mathtt{x} \in \mathcal{S}^{\mathtt{T}^{nat},K,SmAb,can}_{imp_{\mathtt{CS}}}$, por $t_{imp_{\mathtt{CS}}}(\mathtt{x}) := \mathtt{(IMP\text{-}CS\text{-}imp\text{-}}\delta\ \mathtt{x)}$.
- $\hat{t}_{imp_{\mathtt{CS}}}\colon \mathcal{D}^{\mathtt{T}^{nat},K,SmAb,red,can}_{imp_{\mathtt{CS}}} \to \mathcal{D}^{\mathtt{T}^{nat},K,SmAb,can}_{imp_{\mathtt{CS}}}$ definida, para cada $\mathtt{f} \in \mathcal{S}^{\mathtt{T}^{nat},K,SmAb,red,can}_{imp_{\mathtt{CS}}}$, del siguiente modo

$$\begin{aligned}\hat{t}_{imp_{\mathtt{CS}}}(\mathtt{f}) := \mathtt{\#'(lambda\ \ (f)}&\\ \mathtt{(make\text{-}IMP\text{-}CS}&\\ \mathtt{:imp\text{-}}\delta\ \ \mathtt{f}&\\ \mathtt{:imp\text{-}}\eta\ \ \mathtt{\#'c}^{p_{\eta^{SmAb}}}\mathtt{))}&\end{aligned}$$

Las i–transformaciones anteriores son isomorfismos en la categoría de implementaciones de $\mathcal{M}^{\mathtt{T}^{nat},SmAb}$ y, además, son inversos uno del otro. Observar también que las i–transformaciones anteriores no dependen del conjunto graduado K que hayamos fijado como base para los símplices geométricos.

Según hemos visto en la parte algebraica de este ejemplo, el objeto final en la categoría de $\mathtt{CS}_{imp}$-álgebras que son familias de implementaciones de `CS`-álgebras, todavía admite una descripción más sencilla. Hemos denotado por $\mathbb{1}^{K,can,gen}$ a ese álgebra y la simplificación que supone está basada en que, conocido el comportamiento del operador cara sobre los símplices geométricos existe un modo universal de extenderlo a los abstractos. Esto tiene su análogo en el marco de implementación, del que nos ocupamos a continuación.

Así, para cada conjunto graduado K en $\mathcal{U}$, dada una función $\mathtt{f}_{\delta_K}\colon$ `(satisfies naturalp)` $\times$ `(satisfies naturalp)` $\times K \to \mathtt{T}_{smp}$ con dominio $\left\{\mathtt{(i,n,g)} \mid \mathtt{(>\ n\ 0)} \wedge \mathtt{g} \in K^n\right\}$ definida sobre símplices geométricos, existe una forma universal de extenderla a los abstractos, es decir, de definir una función $\mathtt{f}_{\delta}\colon$ `(satisfies naturalp)` $\times$ `(satisfies naturalp)` $\times \mathtt{T}_{smp} \to \mathtt{T}_{smp}$ con dominio $\left\{\left(\mathtt{i},\mathtt{n},\left\langle \mathtt{(}\mathtt{j_k}\ldots\mathtt{j_1}\mathtt{)\ g}\right\rangle\right) \mid \mathtt{(>\ n\ 0)} \wedge \exists\ \mathtt{l} \in \mathtt{(satisfies\ \ naturalp)} \text{ con } \mathtt{g} \in K^l \text{ y } \mathtt{k}+\mathtt{l}=n\right\}$ que coincida con $\mathtt{f}_{\delta_K}$ sobre los símplices geométricos. Para ello nos apoyaremos en las siguientes funciones:

- `aniadirNatALista`: `(satisfies naturalp)` $\times$ `list` $\to$ `list` $\times$ `(satisfies naturalp)` definida sobre las listas de datos de tipo `integer` estrictamente decrecientes o vacías. Su comportamiento es el que resulta de interpretar cada par $(\mathtt{i},\mathtt{(}\mathtt{j_k}\ldots\mathtt{j_1}\mathtt{)}) \in Def(\mathtt{aniadirNatALista})$ como la composición de operadores $\delta_i\eta_{j_k}\ldots\eta_{j_1}$, que aplicando las propiedades de conjunto simplicial transforma el par anterior en un par $(\mathtt{(}\mathtt{l_p}\ldots\mathtt{l_1}\mathtt{)},\mathtt{j})$ de forma que $\delta_i\eta_{j_k}\ldots\eta_{j_1} = \eta_{l_p}\ldots\eta_{l_1}\delta_j$. (Las propiedades que verifican los conjuntos simpliciales garantizan que la función anterior está bien definida.)
- `compose`: `list` $\times$ `list` $\to$ `list` con dominio las listas de datos de tipo `integer` estrictamente decrecientes o vacías y con comportamiento inducido por el comportamiento de los operadores de degeneración. Así, definimos $\mathtt{compose}(\mathtt{(}\mathtt{l_k}\ldots\mathtt{l_1}\mathtt{)},\mathtt{(}\mathtt{m_s}\ldots\mathtt{m_1}\mathtt{)}) := \mathtt{(}\mathtt{n_r}\ldots\mathtt{n_1}\mathtt{)}$ con $\eta_{l_k}\ldots\eta_{l_1}\eta_{m_s}\ldots\eta_{m_1} = \eta_{n_r}\ldots\eta_{n_1}$. (Al igual que en el caso anterior, las propiedades de los conjuntos simpliciales aseguran que la función anterior está bien definida.)

Así, si una terna $\big(\mathtt{i},\mathtt{n},\langle\mathtt{(j_k\ldots j_1)\ g}\rangle\big)$ está en el dominio de $\mathtt{f}_\delta$, conocida la función $\mathtt{f}_{\delta_K}$, para definir $\mathtt{f}_\delta\big(\mathtt{i},\mathtt{n},\langle\mathtt{(j_k\ldots j_1)\ g}\rangle\big)$ el proceso es el siguiente:

- aplicamos $\mathtt{aniadirNatAListA(i,(j_k\ldots j_1))}$ y denotamos al resultado por $(\mathtt{(l_p\ldots l_1)},\mathtt{j})$;
- aplicamos $\mathtt{f}_{\delta_K}(\mathtt{j},\mathtt{n}-\mathtt{k},\mathtt{g})$ y obtenemos un símplice abstracto que denotamos por $\langle\mathtt{(m_s\ldots m_1)\ g'}\rangle$
- componemos a través de la función `compose` las dos listas de degeneraciones obtenidas, es decir, $\mathtt{compose(\ldots)}$, $\mathtt{(m_s\ldots m_1)}$) y obtenemos una lista $\mathtt{(n_r\ldots n_1)}$;
- finalmente, $\mathtt{f}_\delta\big(\mathtt{i},\mathtt{n},\langle\mathtt{(j_k\ldots j_1)\ g}\rangle\big)$ es $\langle\mathtt{(n_r\ldots n_1)\ g'}\rangle$.

Es claro que las dos funciones auxiliares anteriores son calculables y que, por tanto, existen programas que las implementan.

Así, dada una función $\mathtt{f}_{\delta_{gsm}}$ definida para símplices geométricos, denotaremos por $\mathtt{f}_{\delta_{asm}}$ a su extensión tal y como acabamos de describir. Esto nos permite dar otra descripción del objeto final en la categoría de $\mathtt{CS}_{imp}$-álgebras definidas a partir de implementaciones de $Imp^{\underline{\mathtt{T}}^{nat}}(\mathcal{T}^{SmAb}_{\mathtt{CS}})$, objeto que denotamos por $\mathcal{M}^{\underline{\mathtt{T}}^{nat},SmAb,EAT}$ y que definimos como sigue:

$$\mathcal{M}^{\underline{\mathtt{T}}^{nat},SmAb,EAT}_{imp_{\mathtt{CS}}} = \Big\{\ \mathtt{f}_{\delta_{gsm}}\colon(\mathtt{satisfies\ naturalp})\times(\mathtt{satisfies\ naturalp})\times K\to\mathtt{T}_{smp}\ \mid \exists\,\mathcal{I}=\big(\underline{\mathcal{R}},\mathtt{p}^{\mathcal{I}}_{\delta},\mathtt{p}^{\mathcal{I}}_{\eta}\big)\in Obj(Imp^{\underline{\mathtt{T}}^{nat}}(\mathcal{T}^{SmAb}_{\mathtt{CS}}))\text{ tal que }\mathtt{f}_{\delta_{asm}}=F(\mathtt{p}^{\mathcal{I}}_{\delta})\Big\}$$

La operación $imp\_\delta_{\mathcal{M}^{\underline{\mathtt{T}}^{nat},SmAb,EAT}}$ se define como la aplicación de la extensión a los símplices abstractos de la función que es el primer argumento al resto, mientras que la operación $imp\_\eta_{\mathcal{M}^{\underline{\mathtt{T}}^{nat},SmAb,EAT}}$ viene dada por la función η^{SmAb}.

Así, sea K un conjunto graduado en $\mathcal{U}$ y consideramos los conjuntos $\mathtt{S}^{K,\delta}$ y $\mathtt{S}^{K,\eta}$ (definidos en la página 184) como soportes de los programas de las implementaciones a considerar.

Definimos una implementación canónica de $\mathcal{M}^{\underline{\mathtt{T}}^{nat},SmAb,EAT}$ que denotamos por $\mathcal{I}^{\underline{\mathtt{T}}^{nat},K,SmAb,EAT,can}$ del siguiente modo:

- Representaciones literales para los tipos $\mathtt{(satisfies\ naturalp)}$ y $\mathtt{T}_{smp}$.
- Para el género $imp_{\mathtt{CS}}$, tomamos como dominio el tipo Lisp `function`. Como soporte tomamos

$$\mathcal{S}^{\underline{\mathtt{T}}^{nat},K,SmAb,EAT,can}_{imp_{\mathtt{CS}}} = \Big\{\mathtt{x}\in\mathtt{function}\ \mid\ \exists\,\mathcal{I}=\big(\underline{\mathcal{R}},\mathtt{p}^{\mathcal{I}}_{\delta},\mathtt{p}^{\mathcal{I}}_{\eta}\big)\in Obj(Imp^{\underline{\mathtt{T}}^{nat}}(\mathcal{T}^{SmAb}_{\mathtt{CS}}))\ \text{con }\Omega_{\mathtt{CS}}\text{-conjunto }(\mathtt{S}^{K,\delta},\mathtt{S}^{K,\eta})\text{ y tal que }\exists\ \mathtt{p}^{\mathtt{x}}_{\delta_{gsm}}\text{ verificando }\mathtt{(eq\ \ x\ \ c}^{\mathtt{p}^{\mathtt{x}}_{\delta_{gsm}}}\mathtt{)}\text{ con }F(\mathtt{p}^{\mathcal{I}}_{\delta})=\mathtt{f}^{\mathtt{x}}_{\delta_{asm}}\Big\}$$

  donde $\mathtt{f}^{\mathtt{x}}_{\delta_{asm}}$ es la extensión de la función definida por el programa $\mathtt{p}^{\mathtt{x}}_{\delta_{gsm}}$ a $\mathtt{S}^{K,\delta}$, y el soporte de $\mathtt{p}^{\mathtt{x}}_{\delta_{gsm}}$ es, por definición, el conjunto $\big\{(\mathtt{i},\mathtt{n},\mathtt{x})\in\mathtt{S}^{K,\delta}\ \mid\ \mathtt{(eq\ (asm\text{-}dop\ x)\ nil)}\ \big\}$.

La función de abstracción $\alpha^{\mathtt{T}^{nat},K,SmAb,EAT,can}_{imp_{\mathtt{CS}}} : \mathcal{S}^{\mathtt{T}^{nat},K,SmAb,EAT,can}_{imp_{\mathtt{CS}}} \to \mathcal{M}^{\mathtt{T}^{nat},SmAb,EAT}_{imp_{\mathtt{CS}}}$ lleva cada objeto funcional `z` a la función $\mathtt{f}^{\mathtt{z}}_{\delta}$: (`satisfies naturalp`) × (`satisfies naturalp`) × $K \to \mathtt{T}_{smp}$ con dominio $\{(\mathtt{i},\mathtt{n},\mathtt{x}) \in \mathtt{S}^{K,\delta} \mid$ `(eq (asm-dop x) nil)` $\}$ y definida por

$$\mathtt{f}^{\mathtt{z}}_{\delta}(\mathtt{i},\mathtt{n},\mathtt{x}) := \texttt{(funcall z i n (make-asm :dop () :gsm x))}.$$

Como igualdad de representación tomamos la de la abstracción.

- Respecto a los programas que implementan a los operadores, consideramos:
  - el programa $\mathtt{p}^{\mathtt{T}^{nat},K,SmAb,EAT,can}_{imp\,\delta}$ tiene por soporte $\mathcal{S}^{\mathtt{T}^{nat},K,SmAb,EAT,can}_{imp_{\mathtt{CS}}} \times \mathtt{S}^{K,\delta}$; tipos de entrada $\mathcal{D}^{\mathtt{T}^{nat},K,SmAb,EAT,can}_{imp_{\mathtt{CS}}}$, `(satisfies naturalp)`, `(satisfies naturalp)` y $\mathtt{T}_{smp}$; tipo de salida $\mathtt{T}_{smp}$; y, el código asociado es el siguiente objeto funcional

$$\mathtt{c}^{\mathtt{p}^{\mathtt{T}^{nat},K,SmAb,EAT,can}_{imp\,\delta}} \equiv \texttt{\#'(lambda (z i n x) (funcall z i n x))}$$

  - $\mathtt{p}^{\mathtt{T}^{nat},K,SmAb,EAT,can}_{imp\,\eta}$ tiene a $\mathcal{S}^{\mathtt{T}^{nat},K,SmAb,EAT,can}_{imp_{\mathtt{CS}}} \times \mathtt{S}^{K,\eta}$ como soporte; a $\mathcal{D}^{\mathtt{T}^{nat},K,SmAb,EAT,can}_{imp_{\mathtt{CS}}}$, `(satisfies naturalp)`, `(satisfies naturalp)` y $\mathtt{T}_{smp}$ como tipos de entrada; a $\mathtt{T}_{smp}$ como tipo de salida; y, como código el siguiente objeto funcional

$$\mathtt{c}^{\mathtt{p}^{\mathtt{T}^{nat},K,SmAb,EAT,can}_{imp\,\eta}} \equiv \texttt{\#'(lambda (z i n x) (funcall c}^{\mathtt{p}_{\eta SmAb}}\texttt{ i n x))}$$

De nuevo, las $\mathtt{CS}_{imp}$-álgebras $\mathcal{M}^{\mathtt{T}^{nat},SmAb,EAT}$ y $\mathcal{M}^{\mathtt{T}^{nat},SmAb,red}$ son isomorfas en la categoría de $\mathtt{CS}_{imp}$-álgebras definidas a partir de implementaciones de $Imp^{\mathtt{T}^{nat}}(\mathcal{T}^{SmAb}_{\mathtt{CS}})$. Este isomorfismo se extiende a i–transformaciones entre las implementaciones canónicas asociadas a cada una de las álgebras anteriores, que resultan ser isomorfismos. Entre los conjuntos soporte del género $imp_{\mathtt{CS}}$ esas i–transformaciones vienen dadas como sigue

- $t'_{imp_{\mathtt{CS}}} : \mathcal{D}^{\mathtt{T}^{nat},K,SmAb,red,can}_{imp_{\mathtt{CS}}} \to \mathcal{D}^{\mathtt{T}^{nat},K,SmAb,EAT,can}_{imp_{\mathtt{CS}}}$ definida por

$$t'_{imp_{\mathtt{CS}}}(\mathtt{f}) := \texttt{\#'(lambda (i n gsm) (funcall f i n (make-asm :dop () :gsm gsm)))}$$

  para cada $\mathtt{f} \in \mathcal{S}^{\mathtt{T}^{nat},K,SmAb,red,can}_{imp_{\mathtt{CS}}}$.

- $\hat{t}'_{imp_{\mathtt{CS}}} : \mathcal{D}^{\mathtt{T}^{nat},K,SmAb,EAT,can}_{imp_{\mathtt{CS}}} \to \mathcal{D}^{\mathtt{T}^{nat},K,SmAb,red,can}_{imp_{\mathtt{CS}}}$ viene definida por la extensión a los símplices abstractos, tal y como se ha explicado anteriormente, de un operador que trabaja únicamente sobre los geométricos.

De las tres descripciones dadas para el objeto que recoge a las implementaciones de conjuntos simpliciales (con determinadas condiciones), la última es la que más se acerca al modo de trabajo de EAT. Por ello, sus implementaciones canónicas son las que más fielmente recogen la construcción de representaciones en la máquina de conjuntos simpliciales. Así, por ejemplo,

fijado un conjunto graduado K y una implementación $\mathcal{I}^{\underline{\mathtt{T}}^{nat},K,SmAb}$ de $\mathcal{M}^{\underline{\mathtt{T}}^{nat},SmAb}$, tenemos el siguiente paso natural hasta la correspondiente implementación canónica

$$\mathcal{I}^{\underline{\mathtt{T}}^{nat},K,SmAb} \xrightarrow{!\text{final}} \mathcal{I}^{\underline{\mathtt{T}}^{nat},K,SmAb,can} \xrightarrow{t} \mathcal{I}^{\underline{\mathtt{T}}^{nat},K,SmAb,red,can} \xrightarrow{t'} \mathcal{I}^{\underline{\mathtt{T}}^{nat},K,SmAb,EAT,can}$$

Por ejemplo, si fijamos como conjunto graduado $K_{\cup}$, ya utilizado anteriormente (página 168) para representar el conjunto de símplices geométricos de esferas, la i–transformación final de la implementación $\mathcal{I}^{K_{\cup}}$ (página 170) en la implementación $\mathcal{I}^{\underline{\mathtt{T}}^{nat},K,SmAb,EAT,can}$, viene dada por:

$l_{imp_{\mathtt{CS}}}(\mathtt{m}) :=$ `#'(lambda (i n gsm)`
`(funcall` $\mathtt{c}^{K_{\cup}}_{imp\_\eta}$ `m i n (make-asm :dop () :gsm gsm)))`

que resulta de componer el morfismo final dado en la página 186 con las i–transformaciones t y t' definidas anteriormente. Además, si consideramos la implementación $\widetilde{\mathcal{I}}^{K_{\cup}}$, implementación de la misma $\mathtt{CS}_{imp}$-álgebra que $\mathcal{I}^{K_{\cup}}$ (página 171), observamos que la i–transformación anterior también es para ella morfismo final, por lo que se alcanza el mismo código funcional que en el caso de la implementación $\mathcal{I}^{K_{\cup}}$.

La existencia de la i–transformación t es consecuencia de que exista una forma universal de programar un determinado operador, es decir, de disponer de un código que, sobre los soportes de los programas de las implementaciones cuya representación en la máquina pueda construirse en tiempo de ejecución, defina la misma función que los programas de la implementación. En ese caso, es claro que disponiendo de ese código, para representar en la máquina una implementación basta disponer de los códigos de los programas asociados al resto de operaciones. Esto es lo que ocurre en nuestro caso con el operador de degeneración, fijado un patrón para los símplices abstractos. Por otro lado, la existencia de t' tiene que ver con la existencia de una forma universal de definir el comportamiento de una función a partir del de otra que trabaja sobre otro conjunto más reducido de datos. Si se dispone de un método para extender el comportamiento de la segunda, es suficiente con considerar como representación en la máquina de una implementación la función definida sobre el conjunto de datos básicos. Esto es lo que ocurre con el operador cara de un conjunto simplicial, que, a partir de su definición sobre símplices geométricos, se extiende de forma única a los símplices abstractos.

Así, podríamos extraer varias consecuencias del ejemplo. Por una parte, si las implementaciones de Σ-álgebras cuyas representaciones en la máquina potencialmente pueden aparecer en un cálculo de EAT poseen alguna operación de forma que, las funciones definidas por los programas que implementan esas operaciones en todas las implementaciones, son las mismas, es suficiente con disponer de ese programa independientemente. Esto permite obtener otra descripción de la implementación canónica que recoge a las mismas implementaciones y en la que no aparecen los objetos funcionales correspondientes a esas operaciones. La cuestión es clara, si hay una forma universal de programarlas, no hay necesidad de almacenarlas (en el objeto final) ya que no forman parte de la información necesaria para recuperar una implementación.

Por otra parte, si existe una forma universal (siguiendo una determinada estrategia) de definir un operador a partir de otro, es suficiente para recuperar el primero, disponer del segundo y de la estrategia. Esto tiene que ver con la generabilidad de un conjunto de datos a partir de unos datos básicos y de unas operaciones.

En ambas situaciones, lo que se tienen son operadores al margen de la implementación canónica que, junto con ella, permiten recuperar cualquier implementación de las de partida.

En otra línea, observar que es posible que, estando en alguna de las situaciones anteriores, se puedan definir esos operadores que no es necesario que aparezcan explícitamente en la implementación canónica (como componentes de las tuplas) sin depender de los dominios fijados como dominios de definición para los programas de las implementaciones de partida. Por ejemplo, en el caso en el que estamos, cualquier operador definido sobre el dominio de la función η^{SmAb} puede restringirse interpretándolo sobre el subconjunto que convenga al fijar un conjunto graduado. Esto hace que haya distintas implementaciones que, por el morfismo final, lleguen a la misma tupla de códigos. Lo anterior pone de manifiesto la existencia del polimorfismo y la sobrecarga de los objetos funcionales contenidos en los objetos finales y, muestra la idea intuitiva de que un mismo código puede definir distintos programas al considerarlo sobre distintos soportes. Además, como los soportes no están en la máquina, sino en la mente del que está codificando, hay distintos modelos que en la máquina se representan del mismo modo y es el usuario el que tiene que discriminar el caso que está tratando.

3.8.2 Complejos de cadenas

Del mismo modo que hemos detallado en el anterior ejemplo el caso de los conjuntos simpliciales, vamos a dar una implementación que recoja a las implementaciones del TAD $\mathcal{T}_{\mathtt{CC}}$, de los complejos de cadenas, con un determinado $G_{\mathtt{CC}}$-tipo. Recordar que la signatura es

$$\begin{array}{lcccl} suma & : & cmbn \;\; cmbn & \to & cmbn \\ opp & : & cmbn & \to & cmbn \\ zero & : & int & \to & cmbn \\ mult & : & int \;\; cmbn & \to & cmbn \\ dffr & : & cmbn & \to & cmbn \end{array}$$

Para fijar el $G_{\mathtt{CC}}$-tipo con el que nos interesa trabajar, para acercarnos al modo de trabajo de EAT, vamos a apoyarnos en lo desarrollado en la parte algebraica de este ejemplo, concretamente, en la representación de cualquier elemento de un complejo de cadenas como una combinación lineal. Si $G = \{G_p\}_{p\in\mathbb{Z}}$ es un conjunto graduado que contiene en cada grado a los generadores del $\mathbb{Z}$-módulo correspondiente, el siguiente conjunto I^G_{cmbn} recoge la representación de cada elemento del complejo de cadenas como combinación lineal de generadores:

$$I^G_{cmbn} = \Big\{ \langle p, [(t_1, g_1), \ldots, (t_m, g_m)] \rangle \;\mid\; p \in \mathbb{Z},\;\; m \in \mathbb{Z},\;\; m \geq 0,\;\; t_i \in \mathbb{Z} \;\text{ con } t_i \neq 0 \text{ para todo } i = 1, \ldots, m,\; g_i \in G_p \text{ para todo } i = 1, \ldots, m \text{ y } g_i \neq g_j \text{ si } i \neq j \Big\}$$

Este conjunto nos sirve como patrón sintáctico para definir el tipo Lisp que se fijará para el género *cmbn* de las combinaciones lineales. Así, consideramos en primer lugar los siguientes tipos

```
(defstruct  mnm  cff  gnr)
(defstruct  cmb  dgr  lst)
```

Tomamos como tipo $\mathtt{T}_{cmbn}$ aquellos datos de tipo `cmb` tales que el dato almacenado en el campo `dgr` es de tipo `integer` y el almacenado en `lst` es una lista (de tipo `list`) formada por datos de tipo `mnm` tales que el dato almacenado en cada campo `cff` es de tipo `integer` pero no `0` y los almacenados en los campos `gnr` de todos los monomios `mnm` son distintos dos a dos (se entiende que distintos por `eq`, única igualdad natural en $\mathcal{U}$). Intuitivamente estamos pensando en que cada elemento `x` de $\mathtt{T}_{cmbn}$ sea una representación de la combinación de grado el entero representado de forma natural por el `integer (cmb-dgr x)` y formada por tantos monomios como `(length (cmb-lst x))`, compuestos cada uno de ellos por el coeficiente entero trasladado del `integer (mnm-cff (nth n (cmb-lst x)))` y el generador `(mnm-gnr (nth n (cmb-lst x)))`, con `n` $\in$ { `0 ... (- (length (cmb-lst x)) 1)` }. Representando cada elemento `x` de tipo $\mathtt{T}_{cmbn}$ como $\langle \mathtt{p}, [(\mathtt{t}_1, \mathtt{x}_1), \ldots, (\mathtt{t}_\mathtt{m}, \mathtt{x}_\mathtt{m})] \rangle$, el conjunto anterior puede describirse del siguiente modo

$$\mathtt{T}_{cmbn} = \Big\{ \langle \mathtt{p}, [(\mathtt{t}_1, \mathtt{x}_1), \ldots, (\mathtt{t}_\mathtt{m}, \mathtt{x}_\mathtt{m})] \rangle \;\mid\; \mathtt{p} \in \texttt{integer},\ \mathtt{m} \in \texttt{integer},\ \texttt{(>= m 0)},\ \mathtt{t_i} \in \texttt{integer} \text{ con } \texttt{(not (= } \mathtt{t_i}\ \texttt{0))} \text{ para todo } \mathtt{i} = 1, \ldots, \mathtt{m}, \text{ y } \texttt{(not (eq } \mathtt{x_i}\ \mathtt{x_j}\texttt{))} \text{ para todo } \mathtt{i}, \mathtt{j} \in \{1, \ldots, \mathtt{m}\} \text{ con } \texttt{(not (= i j))} \Big\}$$

Hasta ahora hemos determinado los elementos que son de tipo $\mathtt{T}_{cmbn}$, pero para terminar de definir el tipo $\mathtt{T}_{cmbn}$ debemos dar una relación de igualdad entre sus datos. En este caso, apoyándonos en la parte algebraica del ejemplo, es claro que interesa definir una igualdad distinta de la específica que posee como subconjunto de $\mathcal{U}$. Concretamente, debemos tener en cuenta que el orden de los monomios dentro de la lista de una combinación no es relevante, de forma que dos combinaciones que posean los mismos monomios en diferente orden, deben ser iguales en el tipo. Así, completamos la definición del tipo $\mathtt{T}_{cmbn}$ considerando que dos combinaciones son iguales en $\mathtt{T}_{cmbn}$ si son del mismo grado y poseen los mismos monomios. Como igualdad entre monomios se considera la específica de Common Lisp, es decir, `=` entre los coeficientes y `eq` entre los generadores.

Fijamos como $G_{\mathtt{CC}}$-tipo $\underline{\mathtt{T}} = \{\texttt{integer}, \mathtt{T}_{cmbn}\}$.

La $\mathtt{CC}_{imp}$-álgebra $\mathcal{M}^{\underline{\mathtt{T}}}$, final en la categoría de $\mathtt{CC}_{imp}$-álgebras definidas a partir de implementaciones de complejos de cadenas del TAD $\mathcal{T}_{\mathtt{CC}}$, con tipos `integer` y $\mathtt{T}_{cmbn}$, tiene como soportes para los géneros *int* y *cmbn* los tipos `integer` y $\mathtt{T}_{cmbn}$, y para el género $imp_{\mathtt{CC}}$ el siguiente conjunto:

$$\mathcal{M}^{\underline{\mathtt{T}}}_{imp_{\mathtt{CC}}} = \Big\{ (\mathtt{f}_{suma}, \mathtt{f}_{opp}, \mathtt{f}_{zero}, \mathtt{f}_{mult}, \mathtt{f}_{dffr}) \mid \exists\ \mathcal{I} = \big(\underline{\mathcal{R}},\ \mathtt{p}^{\mathcal{I}}_{suma},\ \mathtt{p}^{\mathcal{I}}_{opp},\ \mathtt{p}^{\mathcal{I}}_{zero},\ \mathtt{p}^{\mathcal{I}}_{mult},\ \mathtt{p}^{\mathcal{I}}_{dffr}\big) \in Obj(Imp^{\underline{\mathtt{T}}}(\mathcal{T}_{\mathtt{CC}})) \text{ tal que } \mathtt{f}_\sigma = F(\mathtt{p}^{\mathcal{I}}_\sigma) \text{ para todo } \sigma \in \{suma, opp, zero, mult, dffr\} \Big\}$$

siendo $Imp^{\underline{\mathtt{T}}}(\mathcal{T}_{\mathtt{CC}})$ la categoría de implementaciones de $\mathcal{T}_{\mathtt{CC}}$ con $G_{\mathtt{CC}}$-tipo $\underline{\mathtt{T}}$. Las operaciones vienen dadas como las proyecciones en cada componente.

Para definir implementaciones canónicas de $\mathcal{M}^{\underline{\mathtt{T}}}$ debemos fijar los soportes para los programas de las implementaciones de los complejos de cadenas que van a considerarse. Otra vez, aunque teóricamente tiene sentido fijar cualquier familia de conjuntos como dominios de los

operadores, siempre sujetos a la restricción impuesta por los tipos ya fijados y por las cabeceras de los operadores, en la práctica estos conjuntos van a venir dados de modo implícito. Para cada conjunto graduado $G = \{G_p\}_{p\in\mathbb{Z}}$ en $\mathcal{U}$ consideramos el siguiente conjunto

$$\mathtt{S}^G = \Big\{ \mathtt{x} \in \mathtt{T}_{cmbn} \;\mid\; \texttt{(mnm-gnr (nth j (cmb-lst x)))} \in G_p, \text{ para todo } \mathtt{j} \in \{\mathtt{0}, \ldots, \texttt{(- (length (cmb-lst x)) 1)}\} \text{ siendo } \texttt{(cmb-dgr x)=p} \Big\}$$

Consideraremos solo implementaciones con los siguientes dominios de definición de los programas:

- $\mathtt{S}^{G,suma} = \big\{(\mathtt{x},\mathtt{y}) \in \mathtt{S}^G \times \mathtt{S}^G \;\mid\; \texttt{(= (cmb-dgr x) (cmb-dgr y))}\big\}$
- $\mathtt{S}^{G,opp} = \mathtt{S}^G$
- $\mathtt{S}^{G,zero} = \texttt{integer}$
- $\mathtt{S}^{G,mult} = \texttt{integer} \times \mathtt{S}^G$
- $\mathtt{S}^{G,dffr} = \mathtt{S}^G$

Otra vez, aunque suponga un pequeño abuso, consideraremos que los conjuntos anteriores solo dependen de G, el resto lo fija el patrón que establecen los dominios de las álgebras.

Definimos la implementación canónica de $\mathcal{M}^{\underline{\mathtt{T}}}$ que denotamos por $\mathcal{I}^{\underline{\mathtt{T}},G,can}$, y que, en este caso, viene dada como sigue:

- Para los géneros *int* y *cmbn* se toman las representaciones literales sobre los tipos `integer` y $\mathtt{T}_{cmbn}$, respectivamente.
- Para el género $imp_{\mathtt{CC}}$, la representación $\mathcal{R}^{\underline{\mathtt{T}},G,can}_{imp_{\mathtt{CC}}}$, que apoyándose en el tipo

  ```
  (defstruct IMP-CC imp-suma imp-opp imp-zero imp-mult imp-dffr)
  ```

  toma como dominio aquellos datos de la estructura anterior que satisfacen el predicado `D-can-CC-p` que decide si un dato es del tipo anterior y sus campos son o no objetos funcionales,

  ```
  (defun D-can-CC-p (x)
       (and (typep x 'IMP-CC)
            (functionp (IMP-CC-imp-suma x))
            (functionp (IMP-CC-imp-opp x))
            (functionp (IMP-CC-imp-zero x))
            (functionp (IMP-CC-imp-mult x))
            (functionp (IMP-CC-imp-dffr x))))
  ```

  Como soporte, consideramos el conjunto

$$\mathcal{S}^{\underline{\mathtt{T}},G,can}_{imp_{\mathtt{CC}}} = \Big\{ \mathtt{x} \in \texttt{(satisfies D-can-CC-p)} \mid \exists\, \mathcal{I} = \big(\underline{\mathcal{R}}, (\mathtt{p}^{\mathcal{I}}_{\sigma})_{\sigma\in\{suma,opp,zero,mult,dffr\}}\big) \in Obj(Imp^{\underline{\mathtt{T}}}(\mathcal{T}_{\mathtt{CC}})) \text{ con } \Omega_{\mathtt{CC}}\text{-conjunto definido a partir de } G \text{ y tal que } \texttt{(eq c}^{\mathtt{p}^{\mathcal{I}}_{\sigma}}\texttt{ (IMP-CC-imp-}\sigma\texttt{x))} \text{ para todo } \sigma \in \{suma, opp, zero, mult, dffr\} \Big\}$$

La función de abstracción $\alpha^{\mathtt{T},G,can}_{imp_{\mathtt{CC}}}\colon \mathcal{S}^{\mathtt{T},G,can}_{imp_{\mathtt{CC}}} \to \mathcal{M}^{\mathtt{T}}_{imp_{\mathtt{CC}}}$ lleva cada registro de códigos funcionales a la tupla de funciones, con dominios $(\mathtt{S}^{G,\sigma})_{\sigma\in\{suma,opp,zero,mult,dffr\}}$, definidas por esos códigos. Como igualdad de representación tomamos la de la abstracción.

- Respecto de los programas que implementan a las operaciones, consideramos, para cada $(\sigma\colon g_1\dots g_n \to g) \in \{suma, opp, zero, mult, dffr\}$, el programa $\mathtt{p}^{\mathtt{T},G,can}_{imp\_\sigma}$ con soporte $\mathcal{S}^{\mathtt{T},G,can}_{imp_{\mathtt{CC}}} \times \mathtt{S}^{G,\sigma}$ siendo

$$\mathtt{c}^{\mathtt{p}^{\mathtt{T},G,can}_{imp\_\sigma}} \equiv \mathtt{\#'(lambda\ (x\ d_1 \dots d_n)}\ \mathtt{(funcall\ (IMP\text{-}CC\text{-}imp\text{-}}\sigma\ \mathtt{x)\ d_1 \dots d_n))}$$

La implementación $\mathcal{I}^{\mathtt{T},G,can}$ es objeto final en una subcategoría adecuada de la categoría de implementaciones de $\mathcal{M}^{\mathtt{T}}$.

Ahora bien, en este ejemplo concreto podemos ir un poco más allá del caso general. No se pierde generalidad suponiendo que el conjunto subyacente a cualquier complejo de cadenas siempre puede verse (puede expresarse) como un subconjunto de

$$Cmb = \Big\{ \big\langle p, [(t_1, x_1), \dots, (t_m, x_m)] \big\rangle \ \mid\ p \in \mathbb{Z},\ \ m \in \mathbb{Z},\ \ m \geq 0,\ \ t_i \in \mathbb{Z} \ \text{ con } t_i \neq 0 \\ \text{para todo } i = 1, \dots, m,\ \text{ y } x_i \neq x_j \text{ si } i \neq j \Big\}$$

El tipo $\mathtt{T}_{cmbn}$ es el trasladado a Common Lisp del conjunto Cmb.

Así, nos podemos restringir al TAD $\mathcal{T}^{Cmb}_{\mathtt{CC}} = \langle \mathtt{CC}, \mathcal{C}^{Cmb}(\mathtt{CC})\rangle$ formado por aquellas CC-álgebras del TAD $\mathcal{T}_{\mathtt{CC}}$ cuyo soporte para el género *cmbn* es un subconjunto de Cmb. En definitiva, lo que se hace es fijar solo un patrón sintáctico para los elementos, dejando variar el conjunto de generadores. Por otra parte, en todas las álgebras de $\mathcal{C}^{Cmb}(\mathtt{CC})$ las operaciones *suma*, *opp*, *zero* y *mult* vienen en cierto modo fijadas por la condición de ser $\mathbb{Z}$-módulo libre en cada grado. Esto es, existen unos operadores universales que recogen el comportamiento de estas operaciones, con independencia de cuál sea el conjunto de generadores. Definimos las siguientes operaciones, representando cada elemento de $\mathtt{T}_{cmbn}$ mediante el esquema $\big\langle \mathtt{p}, [(\mathtt{t}_1, \mathtt{x}_1), \dots, (\mathtt{t}_m, \mathtt{x}_m)]\big\rangle$,

- $\mathtt{suma}^{\mathcal{U}}\colon \mathtt{T}_{cmbn} \times \mathtt{T}_{cmbn} \to \mathtt{T}_{cmbn}$, es la función parcial con dominio $\mathtt{S}^{G,suma}$ y que se define "concatenando" los monomios de las dos combinaciones que se suman, sumando entre sí los (coeficientes de) los monomios con el mismo (por `eq`) dato en el campo `gnr` (mismo generador) eliminando el monomio obtenido si el coeficiente es nulo.
- $\mathtt{opp}^{\mathcal{U}}\colon \mathtt{T}_{cmbn} \to \mathtt{T}_{cmbn}$ es la función total definida por

$$\mathtt{opp}^{\mathcal{U}}\big(\big\langle \mathtt{p}, [(\mathtt{t}_1, \mathtt{x}_1), \dots, (\mathtt{t}_m, \mathtt{x}_m)]\big\rangle\big) := \big\langle \mathtt{p}, [(-\mathtt{t}_1, \mathtt{x}_1), \dots, (-\mathtt{t}_m, \mathtt{x}_m)]\big\rangle$$

- $\mathtt{zero}^{\mathcal{U}}\colon \mathtt{integer} \to \mathtt{T}_{cmbn}$ es la función total $\mathtt{zero}^{\mathcal{U}}(\mathtt{p}) := \big\langle \mathtt{p}, [(\ )]\big\rangle$
- $\mathtt{mult}^{\mathcal{U}}\colon \mathtt{integer} \times \mathtt{T}_{cmbn} \to \mathtt{T}_{cmbn}$ es también función total y se define por

$$\mathtt{mult}^{\mathcal{U}}\big(\mathtt{i}, \big\langle \mathtt{p}, [(\mathtt{t}_1, \mathtt{x}_1), \dots, (\mathtt{t}_m, \mathtt{x}_m)]\big\rangle\big) := \big\langle\ \mathtt{p},\ [(\mathtt{(*\ i\ t_1)}, \mathtt{x}_1),\ \dots\ ,\ (\mathtt{(*\ i\ t_m)}, \mathtt{x}_m)]\big\rangle$$

Observando las definiciones anteriores, es claro que las funciones anteriores son calculables. Denotamos por $\mathtt{p}_{\mathtt{suma}^{\mathcal{U}}}$, $\mathtt{p}_{\mathtt{opp}^{\mathcal{U}}}$, $\mathtt{p}_{\mathtt{zero}^{\mathcal{U}}}$ y $\mathtt{p}_{\mathtt{mult}^{\mathcal{U}}}$ a programas que las implementan.

Dado un conjunto graduado $G = \{G_p\}_{p\in\mathbb{Z}}$ en $\mathcal{U}$, nos vamos a limitar a considerar implementaciones de complejos de cadenas modelados por el TAD $\mathcal{T}_{\mathtt{CC}}^{Cmb}$ en las que, además de fijar como $G_{\mathtt{CC}}$-tipo $\underline{\mathtt{T}} = \{\mathtt{integer}, \mathtt{T}_{cmbn}\}$, se va a exigir que la representación del género *cmbn* sea la natural (la abstracción de un elemento de $\mathtt{T}_{cmbn}$ es la correspondiente combinación de Cmb). Denotamos por $Imp^{\underline{\mathtt{T}}^{nat}}(\mathcal{T}_{\mathtt{CC}}^{Cmb})$ a esa subcategoría de implementaciones (plena como subcategoría de $Imp(\mathcal{T}_{\mathtt{CC}}^{Cmb})$). La elección de las representaciones naturales se traduce, por una parte, en que los soportes de los programas de las implementaciones de la categoría anterior serán los subconjuntos de $\mathtt{T}_{cmbn}$ trasladados de forma natural a Common Lisp a partir del trasladado del conjunto graduado G sobre el que se ha definido la implementación; y, por otra, en que las funciones definidas por los programas de estas implementaciones serán las restricciones, a los dominios inducidos por $\mathtt{S}^G$, de los operadores universales definidos anteriormente, es decir, serán $\mathtt{suma}^{\mathcal{U}}|_{\mathtt{S}^{G,suma}}$, $\mathtt{opp}^{\mathcal{U}}|_{\mathtt{S}^{G,opp}}$, $\mathtt{zero}^{\mathcal{U}}|_{\mathtt{S}^{G,zero}}$ y $\mathtt{mult}^{\mathcal{U}}|_{\mathtt{S}^{G,mult}}$. Así, de modo natural, el objeto que según nuestra jerga recoge a las implementaciones de la categoría anterior admite una descripción funcional como $\mathtt{CC}_{imp}$-álgebra, tomando como soportes `integer` y $\mathtt{T}_{cmbn}$ para *int* y *cmbn*, respectivamente, y como soporte para $imp_{\mathtt{CC}}$

$$\mathcal{M}_{imp_{\mathtt{CC}}}^{\underline{\mathtt{T}}^{nat},Cmb,red} = \Big\{ \mathtt{f}_{dffr} \;\mid\; \exists\, \mathcal{I} = \big(\underline{\mathcal{R}}, (\mathtt{p}_\sigma^{\mathcal{I}})_{\sigma\in\{suma,opp,zero,mult,dffr\}}\big) \in Obj(Imp^{\underline{\mathtt{T}}^{nat}}(\mathcal{T}_{\mathtt{CC}}^{Cmb})) \text{ tal que } \mathtt{f}_{dffr} = F(\mathtt{p}_{dffr}^{\mathcal{I}}) \Big\}$$

La operación $imp\_dffr_{\mathcal{M}^{\underline{\mathtt{T}}^{nat},Cmb,red}}$ se define como la aplicación de la función que es el primer argumento al resto, mientras que el resto de operaciones vienen dadas por las funciones $\mathtt{suma}^{\mathcal{U}}$, $\mathtt{opp}^{\mathcal{U}}$, $\mathtt{zero}^{\mathcal{U}}$ y $\mathtt{mult}^{\mathcal{U}}$ restringidas a los dominios correspondientes, dominios proporcionados por los programas de la implementación $\mathcal{I}$.

Desde la perspectiva del Álgebra, el modelo anterior se corresponde con la $\mathtt{CC}_{imp}$-álgebra que denotamos por $\mathbb{1}^{G,can}$, que solo recoge como información de cada $\mathtt{CC}$-álgebra la función correspondiente a la diferencial, ya que las demás están implícitamente determinadas.

Para cada G conjunto graduado en $\mathcal{U}$, fijamos los conjuntos $\mathtt{S}^{G,suma}$, $\mathtt{S}^{G,opp}$, $\mathtt{S}^{G,zero}$ y $\mathtt{S}^{G,mult}$ y denotamos por $\mathcal{I}^{\underline{\mathtt{T}}^{nat},G,Cmb,red,can}$ a la siguiente implementación de $\mathcal{M}^{\underline{\mathtt{T}}^{nat},Cmb,red}$:

- Como representaciones para los tipos `integer` y $\mathtt{T}_{cmbn}$, tomamos las literales.
- Para dar la representación del género $imp_{\mathtt{CC}}$, tomamos como dominio el tipo `function`; como soporte el siguiente conjunto

$$\mathcal{S}_{imp_{\mathtt{CC}}}^{\underline{\mathtt{T}}^{nat},G,Cmb,red,can} = \Big\{ \mathtt{x} \in \mathtt{function} \mid \exists\ \mathcal{I} = \big(\underline{\mathcal{R}}, (\mathtt{p}_\sigma^{\mathcal{I}})_{\sigma\in\{suma,opp,zero,mult,dffr\}}\big) \in Obj(Imp^{\underline{\mathtt{T}}^{nat}}(\mathcal{T}_{\mathtt{CC}}^{Cmb})) \text{ con } \Omega_{\mathtt{CC}}\text{-conjunto } \big(\mathtt{S}^{G,suma}, \mathtt{S}^{G,opp}, \mathtt{S}^{G,zero}, \mathtt{S}^{G,mult}, \mathtt{S}^{G,dffr}\big) \text{ y tal que } \mathtt{(eq\ x\ c}^{\mathtt{p}_{dffr}^{\mathcal{I}}}\mathtt{)} \Big\}$$

  La función de abstracción lleva cada objeto funcional a la función que sobre $\mathtt{S}^{G,dffr}$ define el propio objeto funcional, y como igualdad de representación, consideramos la de la abstracción.

- para cada $\sigma \in \{suma, opp, zero, mult\}$, $\mathtt{p}_{imp\_\sigma}^{\underline{\mathtt{T}}^{nat},G,Cmb,red,can}$ es el programa con soporte $\mathcal{S}_{imp_{\mathtt{CC}}}^{\underline{\mathtt{T}}^{nat},G,Cmb,red,can} \times \mathtt{S}^{G,\sigma}$ y con el siguiente código

$$\mathtt{c}^{\mathtt{p}_{imp\_\sigma}^{\underline{\mathtt{T}}^{nat},G,Cmb,red,can}} \equiv \mathtt{\#'(lambda\ (x\ d_1 \ldots d_n)\ (funcall\ \ c^{p_{\sigma^{\mathcal{U}}}}\ d_1 \ldots d_n))}$$

- $\mathtt{p}_{imp\_diffr}^{\underline{\mathtt{T}}^{nat},G,Cmb,red,can}$ es el programa con soporte $\mathcal{S}_{imp_{\mathtt{CC}}}^{\underline{\mathtt{T}}^{nat},G,Cmb,red,can} \times \mathtt{S}^{G,diffr}$ y siendo el código asociado el siguiente objeto funcional

$$\mathtt{c}^{\mathtt{p}_{imp\_diffr}^{\underline{\mathtt{T}}^{nat},G,Cmb,red,can}} \equiv \mathtt{\#'(lambda\ (x\ cmb)\ (funcall\ x\ cmb))}$$

Esta implementación $\mathcal{I}^{\underline{\mathtt{T}}^{nat},G,Cmb,red,can}$ es isomorfa en $Imp^{\underline{\mathtt{T}}^{nat}}(\mathcal{T}_{\mathtt{CC}}^{Cmb})$ a la implementación $\mathcal{I}^{\underline{\mathtt{T}}^{nat},G,Cmb,can}$ (descripción general del objeto final en esa categoría), por lo que es objeto final.

Volviendo a la parte algebraica de este ejemplo, vimos otra descripción más reducida del objeto que recoge a familias de implementaciones de CC-álgebras, descripción que denotamos por $\mathbb{1}^{G,can,gen}$. La simplificación en la descripción de este objeto respecto al objeto $\mathbb{1}^{G,can}$ proviene de que existe un modo canónico de extender a cualquier elemento de cualquier $\mathbb{Z}$-módulo de un complejo de cadenas el operador diferencial definido únicamente sobre los generadores. En el marco de implementación la situación es la misma.

Así, cada función parcial $\mathtt{df}_{gen}^{\mathcal{U}}\colon \mathtt{integer} \times \mathcal{U} \to \mathtt{T}_{cmbn}$ posee una extensión $\mathtt{df}_{cmbn}^{\mathcal{U}}\colon \mathtt{T}_{cmbn} \to \mathtt{T}_{cmbn}$, definida por linealidad a partir de $\mathtt{df}_{gen}^{\mathcal{U}}$ que viene dada como sigue

$$\mathtt{df}_{cmbn}^{\mathcal{U}}\big(\langle \mathtt{p}, [(\mathtt{t}_1, \mathtt{x}_1), \ldots, (\mathtt{t}_m, \mathtt{x}_m)]\rangle\big) := \mathtt{suma}^{\mathcal{U}}(\ \mathtt{mult}^{\mathcal{U}}(\ \mathtt{t}_1,\ \mathtt{df}_{gen}^{\mathcal{U}}(\mathtt{p}, \mathtt{x}_1)),\ \mathtt{suma}^{\mathcal{U}}(\ \mathtt{mult}^{\mathcal{U}}(\ \mathtt{t}_2,\ \mathtt{df}_{gen}^{\mathcal{U}}(\mathtt{p}, \mathtt{x}_2)),\ \ldots\ \mathtt{suma}^{\mathcal{U}}(\mathtt{mult}^{\mathcal{U}}(\mathtt{t}_m, \mathtt{df}_{gen}^{\mathcal{U}}(\mathtt{p}, \mathtt{x}_m)),\ \mathtt{zero}^{\mathcal{U}}(\mathtt{p}-1))\ldots))$$

Así, un elemento $\langle \mathtt{p}, [(\mathtt{t}_1, \mathtt{x}_1), \ldots, (\mathtt{t}_m, \mathtt{x}_m)]\rangle$ está en el dominio de $\mathtt{df}_{cmbn}^{\mathcal{U}}$ si cada par $(\mathtt{p}, \mathtt{x}_i)$ está en el dominio de $\mathtt{df}_{gen}^{\mathcal{U}}$.

Además, de la propia definición se desprende que si existe un programa que implementa a una función $\mathtt{df}_{gen}^{\mathcal{U}}$, entonces es posible definir un programa que implemente a la función $\mathtt{df}_{cmbn}^{\mathcal{U}}$. De hecho, la definición de $\mathtt{df}_{cmbn}^{\mathcal{U}}$ da un método para hacerlo: el programa construye a partir de una combinación cmb otra combinación de forma que el grado de ésta última es uno menos que el de cmb y la lista de monomios se construye aplicando la diferencial definida sobre los generadores a todos los generadores de la combinación cmb, y multiplicando las combinaciones obtenidas por los coeficientes que han quedado pendientes para finalmente sumar todas las combinaciones obtenidas durante el cálculo.

La forma universal de extender por linealidad cada función parcial de $\mathtt{integer} \times \mathcal{U}$ a $\mathtt{T}_{cmbn}$, también de forma parcial, nos permite dar otra descripción, que denotamos $\mathcal{M}^{\underline{\mathtt{T}}^{nat},Cmb,gen}$, del objeto final en la categoría de $\mathtt{CC}_{imp}$-álgebras que recogen a implementaciones de $Imp^{\underline{\mathtt{T}}^{nat}}(\mathcal{T}_{\mathtt{CC}}^{Cmb})$, descripción que difiere de la del objeto $\mathcal{M}^{\underline{\mathtt{T}}^{nat},Cmb,red}$ en el conjunto soporte para $imp_{\mathtt{CC}}$, que en este caso viene dado como sigue

$$\mathcal{M}_{imp_{\mathtt{CC}}}^{\underline{\mathtt{T}}^{nat},Cmb,gen} = \Big\{ \mathtt{f}_{diffr_{gen}}\colon \mathtt{integer} \times \mathcal{U} \to \mathtt{T}_{cmbn} \ \mid\ \exists\, \mathcal{I} = \big(\underline{\mathcal{R}}, (\mathtt{p}_\sigma^{\mathcal{I}})_{\sigma \in \{suma,opp,zero,mult,diffr\}}\big) \in Obj(Imp^{\underline{\mathtt{T}}^{nat}}(\mathcal{T}_{\mathtt{CC}}^{Cmb}))\ \text{ tal que }\ \mathtt{f}_{diffr_{cmbn}} = F(\mathtt{p}_{diffr}^{\mathcal{I}}) \Big\}$$

donde $\mathtt{f}_{\mathit{diffr}_{cmbn}}$ denota la extensión, por linealidad, de la función $\mathtt{f}_{\mathit{diffr}_{gen}}$.

Así, podemos dar las implementaciones canónicas para este objeto. Para cada conjunto graduado G en $\mathcal{U}$, fijados los conjuntos $\mathtt{S}^{G,suma}$, $\mathtt{S}^{G,opp}$, $\mathtt{S}^{G,zero}$, $\mathtt{S}^{G,mult}$ y $\mathtt{S}^{G,\mathit{diffr}}$, denotamos por $\mathcal{I}^{\underline{\mathtt{T}}^{nat},G,Cmb,gen,can}$ a la siguiente implementación de $\mathcal{M}^{\underline{\mathtt{T}}^{nat},Cmb,gen}$:

- Para los tipos `integer` y $\mathtt{T}_{cmbn}$ consideramos las representaciones literales.
- El tipo `function` como dominio para la representación del género $imp_{\mathtt{CC}}$. Para definir el soporte debemos tener en cuenta que los elementos de éste representan códigos de programas que implementan las diferenciales solo sobre los generadores. Así, tomamos como soporte

$$\mathcal{S}_{imp_{\mathtt{CC}}}^{\underline{\mathtt{T}}^{nat},G,Cmb,gen,can} = \Big\{ \mathtt{x} \in \texttt{function} \mid \exists\ \mathcal{I} = \big(\underline{\mathcal{R}}, (\mathtt{p}_\sigma^{\mathcal{I}})_{\sigma\in\{suma,opp,zero,mult,\mathit{diffr}\}}\big) \in Obj(Imp^{\underline{\mathtt{T}}^{nat}}(\mathcal{T}_{\mathtt{CC}}^{Cmb})) \text{ con } \Omega_{\mathtt{CC}}\text{-conjunto } \big(\mathtt{S}^{G,suma}, \mathtt{S}^{G,opp}, \mathtt{S}^{G,zero}, \mathtt{S}^{G,mult}, \mathtt{S}^{G,\mathit{diffr}}\big) \text{ y tal que existe un programa } \mathtt{p}_{\mathit{df},gen}^{\mathtt{x}} \text{ verificando } \texttt{(eq x } \mathtt{c}^{\mathtt{p}_{\mathit{df},gen}^{\mathtt{x}}}\texttt{)} \text{ con } F(\mathtt{p}_{\mathit{diffr}}^{\mathcal{I}}) = F(\mathtt{p}_{\mathit{df},cmbn}^{\mathtt{x}}) \Big\}$$

  donde $F(\mathtt{p}_{\mathit{df},cmbn}^{\mathtt{x}})$ denota la extensión por linealidad de la función definida por el programa $\mathtt{p}_{\mathit{df},gen}^{\mathtt{x}}$ a $\mathtt{S}^{G,\mathit{diffr}}$, soporte de $F(\mathtt{p}_{\mathit{diffr}}^{\mathcal{I}})$. Observar que, por definición, el soporte de $\mathtt{p}_{\mathit{df},gen}^{\mathtt{x}}$ es el conjunto $\{(\mathtt{p},\mathtt{g}) \mid \mathtt{g} \in G_p\}$.

  Cada $\mathtt{x} \in \mathcal{S}_{imp_{\mathtt{CC}}}^{\underline{\mathtt{T}}^{nat},G,Cmb,gen,can}$, es llevado, por la función de abstracción, a la función parcial $\mathtt{f}_{gen}^{\mathtt{x}}\colon \texttt{integer} \times G \to \mathtt{T}_{cmbn}$ con dominio $\{(\mathtt{p},\mathtt{g}) \mid \mathtt{g} \in G_p\}$ y definida por $\mathtt{f}_{gen}^{\mathtt{x}}(\mathtt{p},\mathtt{g}) := \texttt{(funcall x p g)}$. Como igualdad de representación consideramos la de la abstracción.
- Para cada $\sigma \in \{suma, opp, zero, mult\}$, $\mathtt{p}_{\mathit{imp\_}\sigma}^{\underline{\mathtt{T}}^{nat},G,Cmb,gen,can}$ es el programa con soporte $\mathcal{S}_{imp_{\mathtt{CC}}}^{\underline{\mathtt{T}}^{nat},G,Cmb,gen,can} \times \mathtt{S}^{G,\sigma}$ y con el siguiente código

$$\mathtt{c}^{\mathtt{p}_{\mathit{imp\_}\sigma}^{\underline{\mathtt{T}}^{nat},G,Cmb,gen,can}} \equiv \texttt{\#'(lambda (x } \mathtt{d}_1 \ldots \mathtt{d}_n\texttt{)}\ \texttt{(funcall } \mathtt{c}^{\mathtt{p}_{\sigma^{\mathcal{U}}}} \mathtt{d}_1 \ldots \mathtt{d}_n\texttt{))}$$

- El programa $\mathtt{p}_{\mathit{imp\_diffr}}^{\underline{\mathtt{T}}^{nat},G,Cmb,gen,can}$ tiene como soporte $\mathcal{S}_{imp_{\mathtt{CC}}}^{\underline{\mathtt{T}}^{nat},G,Cmb,gen,can} \times \mathtt{S}^{G,\mathit{diffr}}$ y un código

$$\mathtt{c}^{\mathtt{p}_{\mathit{imp\_diffr}}^{\underline{\mathtt{T}}^{nat},G,Cmb,gen,can}} \equiv \texttt{\#'(lambda (x cmb) ...)}$$

  en el que se extiende por linealidad el comportamiento del objeto funcional `x`.

Lo mismo que nos planteamos en el caso de los conjuntos simpliciales, para los complejos de cadenas se tiene el isomorfismo entre las álgebras $\mathcal{M}^{\underline{\mathtt{T}}^{nat},Cmb,red}$ y $\mathcal{M}^{\underline{\mathtt{T}}^{nat},Cmb,gen}$ que se extiende a un isomorfismo en el nivel de las implementaciones canónicas. Para cada conjunto graduado $G = \{G_p\}_{p\in\mathbb{Z}}$ las siguientes i–transformaciones definen el isomorfismo:

- $t'_{imp_{\mathtt{CC}}}\colon \mathcal{D}_{imp_{\mathtt{CC}}}^{\mathtt{T}^{nat},G,Cmb,red,can} \to \mathcal{D}_{imp_{\mathtt{CC}}}^{\mathtt{T}^{nat},G,Cmb,gen,can}$ definida, para cada $\mathtt{f} \in \mathcal{S}_{imp_{\mathtt{CC}}}^{\mathtt{T}^{nat},G,Cmb,red,can}$, por

```
t'impCC(f) := #'(lambda  (p  gnr)
                   (funcall  f
                        (make-cmb
                             :dgr  p
                             :lst  (list  (make-mnm
                                               :cff  1
                                               :gnr  gnr)))))
```

- $\hat{t}'_{imp_{\mathtt{CC}}}\colon \mathcal{D}_{imp_{\mathtt{CC}}}^{\mathtt{T}^{nat},G,Cmb,gen,can} \to \mathcal{D}_{imp_{\mathtt{CC}}}^{\mathtt{T}^{nat},G,Cmb,red,can}$ definida extendiendo por linealidad cada elemento del soporte, tal y como se ha explicado anteriormente.

De las descripciones vistas para el objeto final, la última de ellas es la más próxima a la que se maneja en EAT.

3.8.3 Algunas conclusiones

Recogemos a continuación algunos comentarios motivados por lo visto en los ejemplos anteriores.

El primero es que la principal característica de las implementaciones canónicas es su naturaleza funcional, coherente con la descripción como tuplas de funciones del objeto final de la parte algebraica. En el marco de implementación esto se traduce en que la mínima información para trabajar en la máquina con un álgebra es un conjunto de objetos funcionales, cada uno de los cuales recoge el comportamiento de un programa que implementa a la operación correspondiente del álgebra representada. En lo anterior, debe tenerse en cuenta que el paso de abstracción no está recogido en la máquina sino que es el usuario el que lo debe tener presente en su mente.

En el estudio de la parte algebraica vimos que, bajo determinadas condiciones, es posible dar descripciones más simples del álgebra final y esto, interpretado en el marco de implementación, se traduce en que puede simplificarse la representación en la máquina de las álgebras.

Concretamente, en los ejemplos anteriores hemos trasladado al marco de implementación las simplificaciones en la descripción del objeto final vistas en la parte algebraica. Recordar que una de ellas corresponde al caso en el que el comportamiento de los programas que implementan a un operador se determina de modo canónico para todas las representaciones de las álgebras con las que se trabaja debido, por un lado al comportamiento de las álgebras respecto a ese operador, y, por otro, a la interpretación (abstracción) que estamos considerando.

La segunda simplificación en la descripción de la implementación final radica en la existencia de una forma universal de extender un operador. Disponiendo de una estrategia común a todas las implementaciones representadas que permita determinar el comportamiento de un programa a partir del comportamiento de otro, es suficiente con almacenar el código del segundo. En los dos ejemplos estudiados, conjuntos simpliciales y complejos de cadenas, son distintas las causas por las que los operadores pueden extenderse, algo que ya comentamos. En el caso de los conjuntos simpliciales, los operadores cara y degeneración pueden extenderse debido al tipo de representación utilizada para los símplices (por una parte, apoyándonos en su representación

como abstractos y, por otra, al codificarlos de modo natural). En el caso de los complejos de cadenas, son, fundamentalmente las propiedades matemáticas de las álgebras las que permiten definir la estrategia para extender los operadores (de los generadores a cualquier combinación). En el desarrollo del ejemplo hemos extendido únicamente la diferencial pero de modo análogo podríamos extender el resto de operaciones (fijado un conjunto graduado G, en lugar de partir de las funciones $\texttt{suma}^{\mathcal{U}}$, $\texttt{opp}^{\mathcal{U}}$, $\texttt{zero}^{\mathcal{U}}$ y $\texttt{mult}^{\mathcal{U}}$, podríamos partir de funciones con aridades $\texttt{integer} \times G \times G$ en $\texttt{T}_{cmbn}$, $\texttt{integer} \times G$ en $\texttt{T}_{cmbn}$, $\texttt{integer}$ en $\texttt{T}_{cmbn}$ e $\texttt{integer} \times \texttt{integer} \times G$ en $\texttt{T}_{cmbn}$, de forma que las extensiones de estas funciones por linealidad correspondan con las funciones $\texttt{suma}^{\mathcal{U}}$, $\texttt{opp}^{\mathcal{U}}$, $\texttt{zero}^{\mathcal{U}}$ y $\texttt{mult}^{\mathcal{U}}$). Así, una descripción más reducida de la implementación final es la descripción de la canónica a partir de un espacio de funciones que definen el comportamiento de las diferenciales sobre los generadores, y teniendo como programas que implementan a las operaciones visibles, por un lado, los programas que trabajan solo sobre generadores y, por otro, los que se apoyan en los anteriores para definir las extensiones por linealidad. En la práctica, en ocasiones resulta más cómodo disponer directamente de los programas que definen las funciones extendidas (en EAT nos encontramos con algunos de ellos).

Otras dos cuestiones a comentar es que hemos fijado representaciones y además, hemos tomado las naturales. De estas dos cuestiones, la esencial es el haber fijado un patrón común de abstracción para las representaciones consideradas. Esto es natural teniendo en cuenta que disponemos de un mismo patrón para los elementos de todos los conjuntos simpliciales o complejos de cadenas a representar, y que, por tanto, es natural que el paso de abstracción para todos ellos sea el mismo, en el sentido de que al trabajar con cualquiera de ellas, el paso de decodificación para llegar al álgebra correspondiente sea el mismo. El hecho de haber considerado como representaciones las naturales nos ha permitido en estos casos trasladar más directamente a las implementaciones las propiedades matemáticas de las álgebras. Considerando otro patrón para las funciones de abstracción, las propiedades matemáticas de las álgebras tendrán otra traducción menos literal en las implementaciones.

El último comentario ilustra el hecho de que la representación de un álgebra no está recogida en lo que hemos denominado representación en la máquina de una implementación. Pensando en considerar funciones de abstracción que sigan un patrón común, incluir como representación en la máquina de una implementación información sobre el soporte y la igualdad de las representaciones, nos permitirá recoger en la máquina más información sobre las implementaciones, de forma que casos que no han quedado recogidos en este marco puedan formalizarse. Por ejemplo, volviendo al ejemplo de los complejos de cadenas, la función definida por cualquier programa que implemente a la operación *suma* debe operar entre sí los monomios que tengan generadores "iguales", en el sentido de que su decodificación por la función de abstracción sea la misma (incluso teniendo en cuenta que sobre ellos puede venir definida explícitamente una relación de igualdad). Debido a que la representación en la máquina de un complejo de cadenas no recoge ninguna información acerca de la representación, en particular no recoge ninguna relación de igualdad entre generadores (la única igualdad entre ellos que puede considerarse para definir la función es la igualdad `eq`) por lo que hay casos que no quedan recogidos. En algunos casos la igualdad definida sobre el conjunto de generadores debe ser incluida en la representación en la máquina, y es aquí donde aparece la necesidad de representar explícitamente la igualdad. En el siguiente capítulo nos ocupamos de ello, concretamente de enriquecer la representación en la máquina de un álgebra incluyendo en ella parte de la representación.

Capítulo 4

Objetos efectivos y localmente efectivos en EAT

4.1 Introducción

En el capítulo anterior hemos establecido un marco formal que nos ha permitido justificar una de las principales características de algunos de los programas de EAT, la posibilidad de construir, en tiempo de ejecución, datos que son representaciones en la máquina de estructuras algebraicas. Sin embargo, si observamos los ejemplos reales de cálculo de EAT mostrados en la sección 1.1, podemos ver que lo hecho hasta ahora no recoge fielmente la representación de cada dato, en el sentido de que no representa completamente la estrategia utilizada por EAT, aunque recoge la parte esencial de ésta (el hecho de representar las álgebras por tuplas de funciones). En la construcción del capítulo anterior se recoge la parte funcional de las operaciones del álgebra pero quedan fuera tres cuestiones. Por una parte, queda pendiente analizar la incorporación de un campo funcional que no hemos considerado en el capítulo anterior, campo éste que incorpora una relación de igualdad. Por otra, también debemos analizar la incorporación a las estructuras de datos de información relevante, de naturaleza no funcional, sobre el álgebra a la que representan. Esta segunda cuestión tiene que ver con lo efectivo y lo localmente efectivo. Finalmente, algunos ejemplares de algunas estructuras de datos son mixtos, en el sentido de que recogen representaciones de objetos efectivos y de objetos localmente efectivos.

En este capítulo nos planteamos dos objetivos. Por una parte, enriquecer el marco establecido en el anterior adecuándolo totalmente a las estructuras de datos de EAT. Por otra, caracterizar la diferencia entre las estructuras mixtas de las que son de naturaleza únicamente funcional. En este sentido, diferenciar lo efectivo de lo localmente efectivo.

Recordar que el marco formal que hemos establecido es una determinada subcategoría de la categoría de implementaciones de un modelo, el cual recoge las posibles representaciones en la máquina de las estructuras algebraicas que pueden aparecer durante el proceso de cálculo. Una forma de describir matemáticamente este modelo es como un espacio funcional, recogiendo cada uno de sus elementos las operaciones de un álgebra de la signatura de partida, luego, todas son álgebras para la misma signatura. Pensando en almacenar en la máquina una representación de este modelo, la descripción anterior corresponde a la representación de los elementos del modelo como tuplas de objetos funcionales, descripción que corresponde con la de las estructuras de datos utilizadas por los algoritmos de cálculo de EAT.

Vamos en primer lugar a ocuparnos de ajustar completamente a EAT el marco introducido en el capítulo anterior. En este sentido, lo enriqueceremos para recoger exactamente el modo de trabajo de EAT, consiguiendo además relajar en parte algunas de las condiciones impuestas.

La noción en la que reposa toda la formalización del capítulo anterior es la noción de implementación. Una implementación de una Σ-álgebra podemos verla como un par: representación de los conjuntos soporte, programas que implementan a las operaciones. Tal y como hemos trabajado en el capítulo anterior, la representación en la máquina de una implementación solo recoge la parte de las operaciones puesto que consta exclusivamente de los objetos funcionales de los programas que implementan las operaciones de las Σ-álgebras. Así, las representaciones de los conjuntos soporte, que constituyen la parte asociada a los géneros de la signatura, han quedado sin considerar. Hemos representado implementaciones pero no las representaciones sobre las que éstas reposan ya que las implementaciones tienen que ver con las operaciones, con Ω, mientras que las representaciones tienen que ver con los géneros. Vamos en este capítulo a ampliar el marco definido en el capítulo anterior representando también las representaciones, es decir, incluyendo la representación de los conjuntos soporte de las Σ-álgebras como parte de la representación en la máquina de una implementación. Representar representaciones se hace habitualmente en determinados campos de Matemáticas e Informática y en nuestro caso concreto es aún más natural, si cabe, si pensamos que algunos de los programas de cálculo construyen representaciones.

La primera cuestión que nos planteamos es qué hay que considerar para "representar representaciones", es decir, cómo representar representaciones. Para responder a lo anterior debemos apoyarnos en la noción de implementación que hemos adoptado, basada en la de representación introducida por Hoare en [58]. Según ésta, una representación consta de una función de abstracción que lleva los datos concretos a los abstractos que representan, de un invariante que determina el dominio de la función de abstracción y de una relación de equivalencia, que interpretamos como una igualdad, sobre los datos del dominio de la función de abstracción. De los tres ingredientes anteriores, evidentemente la función de abstracción no puede ser representada en el computador, así para representar una representación consideraremos un invariante que determinará los datos concretos que representan a alguno abstracto y una relación de igualdad entre los datos concretos considerados. Por tanto, una forma de representar una representación es a través de un par invariante-igualdad.

Desde otras perspectivas también es adecuado considerar al par invariante-igualdad como representación de una representación. Una de ellas viene de observar que las representaciones están relacionadas con los géneros de una signatura y no con las operaciones. Un género es la abstracción de un tipo en un lenguaje de programación (no abstracto), por ejemplo, definido a través del `TYPE` de Pascal. Reynolds, en [93] (página 327 y siguientes), aborda el estudio teórico de los tipos de los lenguajes de programación a través de lo que denomina semánticas extrínsecas o externas. Para Reynolds, una semántica extrínseca es una *Relación de Equivalencia Parcial (PER)*. Tomando como universo el universo Common Lisp $\mathcal{U}$, un *PER* es un subconjunto de $\mathcal{U}$ junto con una relación de equivalencia sobre el subconjunto. Así, especificando algebraicamente, un PER es un par formado por una función total $inv\colon \mathcal{U} \to Bool$ que determina un subconjunto de $\mathcal{U}$ y una función parcial $eql\colon \mathcal{U} \times \mathcal{U} \to Bool$ con dominio el producto por sí mismo del subconjunto de $\mathcal{U}$ determinado por la función inv. Apoyándonos en lo anterior, representar un conjunto sobre un tipo concreto Common Lisp `T` coincide con implementar un PER sobre `T`.

En otra línea dentro de las Matemáticas, la noción de conjunto en Matemática Constructiva coincide con lo que hemos definido como representación de un conjunto. En Matemática Constructiva, un conjunto está dotado, por definición, de una relación de equivalencia interpre-

tada como una igualdad entre los elementos del conjunto (véase [15], página 13). Luego, un conjunto en Matemática Constructiva es un par invariante-igualdad. Desde el punto de vista de la Informática, ver un conjunto como el par invariante-igualdad corresponde con la noción de codificación funcional de Sergeraert [113], imprescindible en el diseño de EAT.

Teniendo en mente el marco establecido en el capítulo anterior, concretamente su parte técnica, incluimos otros comentarios que justifican el añadir como parte de la representación en la máquina de un álgebra el invariante y la igualdad de la representación. Por un lado, éstos aparecen en EAT (en su versión final no aparece el invariante pero por motivos de eficiencia). Por otro, buscando la completitud, en el sentido de que hay casos con los que EAT trabaja y que no están recogidos en lo desarrollado hasta ahora, debido a la rigidez del marco establecido en algunos aspectos.

Para definirlo hemos impuesto determinadas condiciones, algunas de ellas naturales y todas ellas justificadas, unas desde el punto de vista del modelado y otras desde el punto de vista operacional. Por ejemplo, para los géneros de las álgebras cuyas representaciones son construidas, se toman representaciones literales. Como ya hemos explicado, esta condición es natural ya que, son datos que se representan a sí mismos. Sin embargo, si lo pensamos desde el punto de vista de lo que pretendemos modelar, se han perdido las representaciones de los conjuntos soporte de las álgebras. Así, el conjunto de condiciones impuestas al definir el marco hace que situaciones con las que EAT trabaja no queden recogidas hasta ahora.

Por ejemplo, un caso que no queda recogido es el trabajo con la $\mathtt{CS}_{imp}$-álgebra que hemos introducido en la página 106 y que hemos denominado Δ, $\mathtt{CS}_{imp}$-álgebra que representa a la familia de complejos simpliciales $\{\Delta^m\}_{m\in\mathbb{N}}$: los dominios de definición de sus operaciones no factorizan (como cubos) en la componente correspondiente al género distinguido $imp_{\mathtt{CS}}$. Buscando otra forma, menos natural, de representar la familia de álgebras $\{\Delta^m\}_{m\in\mathbb{N}}$, nos hemos apoyado en una $\mathtt{CS}$-álgebra que hemos denominado Δ^∞, caso que tampoco queda recogido porque los dominios de definición de las operaciones son distintos para cada álgebra Δ^m.

Vamos a desarrollar a continuación un ejemplo completo, y más sencillo, para ver más claramente lo anterior. Suponer un algoritmo de cálculo que sabemos debe construir algún grupo finito $\mathbb{Z}_n = \mathbb{Z}/n\mathbb{Z}$ con $n > 1$. El marco formal en el que estudiar esta situación es alguna categoría con objetos las familias de grupos finitos, marco que podemos obtener aplicando la operación $()_{imp}$ a la categoría de grupos finitos. Desde el punto de vista de implementación, esto corresponde a trabajar con familias de representaciones en la máquina de grupos finitos abelianos. Así, en este ejemplo, el punto de partida es el TAD $\mathcal{T}_{\mathtt{GRP}} = \langle \mathtt{GRP}, \mathcal{C}(\mathtt{GRP})\rangle$ definido en el ejemplo 3.7.4 del capítulo anterior, siendo $\mathtt{GRP}$ la signatura

$$\begin{array}{lcccc} prd & : & g\ g & \to & g \\ inv & : & g & \to & g \\ unt & : & & \to & g \end{array}$$

y siendo $\mathcal{C}(\mathtt{GRP})$ la subcategoría plena de la categoría de grupos con objetos los grupos cociente de $\mathbb{Z}$ módulo n, para $n > 1$.

Para cada $n > 1$, definimos una implementación de la $\mathtt{GRP}$-álgebra $\langle \mathbb{Z}/n\mathbb{Z}, \{+_{\mathbb{Z}/n\mathbb{Z}}, -_{\mathbb{Z}/n\mathbb{Z}}, 0_{\mathbb{Z}/n\mathbb{Z}}\}\rangle$, implementación que denotamos por $\mathcal{I}^n_{\mathtt{GRP}}$, en la que los objetos Lisp

que representan a los elementos del conjunto $\mathbb{Z}/n\mathbb{Z}$ son los datos `0`, ... , `n-1` del tipo `integer`:

- como representación del conjunto $\mathbb{Z}/n\mathbb{Z}$ tomamos la representación con dominio `integer`, soporte el conjunto `(mod n)` y función de abstracción la que a cada `i` entero Common Lisp le asocia la clase del entero de las Matemáticas i, representación dada en el ejemplo 3.4.4 del capítulo anterior;
- como programas que implementan a las operaciones consideramos los siguientes:
  - $\mathtt{p}^n_{prd} = (\mathtt{c}^{\mathtt{p}^n_{prd}}, \mathtt{(mod\ n)} \times \mathtt{(mod\ n)}, \mathtt{integer}, \mathtt{integer}, \mathtt{integer})$ siendo
    $\mathtt{c}^{\mathtt{p}^n_{prd}} \equiv \mathtt{\#'(lambda\ (x\ y)\ (mod\ (+\ x\ y)\ n))}$
  - $\mathtt{p}^n_{inv} = (\mathtt{c}^{\mathtt{p}^n_{inv}}, \mathtt{(mod\ n)}, \mathtt{integer}, \mathtt{integer})$ con
    $\mathtt{c}^{\mathtt{p}^n_{inv}} \equiv \mathtt{\#'(lambda\ (x)\ (mod\ (-\ x)\ n))}$
  - $\mathtt{p}^n_{unt} = (\mathtt{c}^{\mathtt{p}^n_{unt}}, \mathtt{nil}, \mathtt{nil}, \mathtt{integer})$ con
    $\mathtt{c}^{\mathtt{p}^n_{unt}} \equiv \mathtt{\#'(lambda\ (\ \ )\ 0)}$

Observando los objetos funcionales de las implementaciones anteriores es natural definir una implementación que recoja a toda la familia, naturalmente será una implementación de una $\mathtt{GRP}_{imp}$-álgebra. Fijándonos en un operador concreto de la signatura anterior y en los correspondientes objetos funcionales de los programas de las implementaciones de la familia anterior, es evidente que podemos construir un objeto funcional, con un argumento más que los de partida y cuya adjunta exponencial en ese argumento nos lleva al objeto funcional de partida. Por ejemplo, para el operador *prd*, el siguiente objeto funcional recoge los objetos funcionales correspondientes a la operación *prd* de las implementaciones anteriores, es decir, es un posible código de un programa que implemente a la operación imp_prd: $imp_{\mathtt{GRP}}\ g\ g \to g$ sobre la familia anterior

$$\mathtt{c}^{\mathtt{p}_{imp\_prd}} \equiv \mathtt{\#'(lambda\ (n\ x\ y)\ (mod\ (+\ x\ y)\ n))}$$

Para continuar debemos asociar dominios de definición a los objetos funcionales para definir programas que codifiquen a las operaciones de la signatura `GRP`. Estos dominios deben factorizar en la componente asociada al género distinguido. Con estas restricciones y habiendo fijado `integer` como tipo para el género correspondiente a los elementos de los grupos, para asociar un dominio de definición al objeto funcional $\mathtt{c}^{\mathtt{p}_{imp\_prd}}$, cualquier posibilidad pasa por asociar `(satisfies naturalp)` $\setminus\{1\} \times$ `integer` $\times$ `integer`. Al hacerlo así, realmente no estamos recogiendo las representaciones que pretendíamos, sino que hemos recogido cualquiera cuyo soporte esté formado por todo el tipo `integer`. De esta forma, en este marco no es posible recoger la familia de implementaciones que pretendíamos, puesto que no tienen el mismo soporte. Como ya se ha comentado, el tipo fijado para un género, en este caso `integer`, proporciona el patrón sintáctico que van a seguir todos los datos, pero es habitual, como ocurre en este caso, que no todos los datos que responden a ese patrón sean significativos, y son las distintas componentes de una representación (soporte, función de abstracción e igualdad de representación) las que permiten acotar los elementos significativos dentro de los que siguen el patrón fijado por el tipo. Así, continuando con el mismo ejemplo, para simular en la máquina los elementos de $\mathbb{Z}/n\mathbb{Z}$ tenemos distintas posibilidades:

1) considerar como soporte de la representación el conjunto `(mod n)`, lo que equivale a considerar cada `i` $\in \{0, \ldots, n-1\}$ como dato que representa a la clase de equivalencia $[i]$;

2) considerar todos los datos de `integer` como significativos y que sea la función de abstracción la que identifica los datos de `integer` que son iguales módulo `n`, es decir, que pertenecen a la misma clase de equivalencia de $\mathbb{Z}/n\mathbb{Z}$;

3) considerar una igualdad de representación que identifique los datos de tipo `integer` que son iguales módulo `n`.

Entre estas tres posibilidades la única que encaja en el marco definido es la que considera como significativos todos los datos de `integer`, teniendo en cuenta que de este modo la parte de representación no queda recogida. La primera posibilidad plantea el problema de no poder fijar un dominio de los operadores común a todas las implementaciones de la familia. La tercera es la que intentamos recoger a continuación.

El ejemplo anterior muestra que para construir implementaciones sobre representaciones no literales debemos recoger en la representación en la máquina de las implementaciones la representación sobre la que reposan. Al trabajar con familias de implementaciones, la representación considerada para los géneros que provienen de la signatura de partida es la literal, representación que no recoge ninguno de los ingredientes del paso del modelo a la máquina, así, ni el soporte, ni la función de abstracción ni la igualdad de la representación han quedado reflejadas en lo que es la representación en la máquina de cada implementación. Por ello, y volviendo al ejemplo anterior, pese a ser natural considerar para cada `n` $\in$ `(satisfies naturalp)`, los datos $\{0, \ldots, n-1\}$ como significativos, este caso no queda cubierto con lo que tenemos hasta ahora ya que en esta situación los dominios de definición de las operaciones son dependientes de cada índice `n` como hemos mostrado en el desarrollo del ejemplo.

En la siguiente sección, ampliamos el marco del capítulo anterior para recoger como representación en la máquina de una implementación, no solo los objetos funcionales de los programas que implementan a las operaciones sino también parte de la representación. Es evidente que si el modelo que estamos representando no "vive" en Common Lisp, el paso del modelo a la máquina no vamos a poder recogerlo en lo que denominamos representación en la máquina de una implementación, sin embargo la existencia de las representaciones naturales (que suponen un mero paso del modelo a la máquina) nos permite identificar los objetos Common Lisp con los datos del álgebra a la que representan y pensar que hemos recogido en la máquina toda la información sobre la implementación del álgebra. Por ello, podemos decir que en el capítulo anterior trabajando exclusivamente con representaciones naturales estábamos incluyendo ya la parte de representación. Sin embargo, en el caso de trabajar con implementaciones sobre representaciones no naturales debemos ampliar lo que entendemos por representación en la máquina de una implementación para recoger el paso al modelo (teniendo en mente siempre un último paso de representación natural que evidentemente no podrá ser recogido si el modelo no está en Common Lisp). Como hemos visto, existen múltiples puntos de vista que nos llevan a enriquecer lo que hemos llamado representación en la máquina de una implementación, incluyendo una parte de la representación sobre la que subyace la implementación formada por un par invariante-igualdad, definiendo el invariante un subconjunto del tipo y estando la igualdad definida sobre ese subconjunto.

4.2 Objetos localmente efectivos

En lo que sigue, el punto de partida es un TAD $\mathcal{T} = \langle \Sigma, \mathcal{C}(\Sigma) \rangle$ formado por una signatura Σ que aporta la parte sintáctica y una categoría de Σ-álgebras $\mathcal{C}(\Sigma)$ que recoge a todas las Σ-álgebras cuyas representaciones pueden aparecer en el proceso de cálculo. Como ya se ha comentado, lo que se construye en tiempo de ejecución son representaciones en la máquina de Σ-álgebras de la categoría $\mathcal{C}(\Sigma)$. Para que esto sea posible, las representaciones en la máquina de las Σ-álgebras deben ser datos, siendo la signatura que modela el trabajo con esos datos la signatura Σ_{imp}, obtenida a partir de Σ aplicando la operación $()_{imp}$. Lo que ha permitido explicar la capacidad de EAT para construir estas representaciones es la existencia de un objeto especial, en una categoría de implementaciones de una Σ_{imp}-álgebra, que recoge a todas las posibles familias de representaciones en la máquina que pueden aparecer en los cálculos. El objeto al que nos referimos es el objeto final de dicha categoría y la cuestión clave es que una de sus descripciones, concretamente aquella en la que los datos de las representaciones en la máquina de las álgebras son tuplas de objetos funcionales, se aproxima a la utilizada por EAT para codificar los datos que manipula. Precisamente, una de las diferencias entre los datos que EAT manipula y la representación en la máquina de un álgebra es que en esta última no hemos recogido parte de la representación. Por ello, vamos a enriquecerla incluyendo parte de la representación sobre la que reposa la implementación. Haciéndolo así veremos que es posible obtener información local de cada implementación, pero no información de naturaleza global, de ahí la denominación de objetos localmente efectivos.

En esta sección nos apoyamos en lo desarrollado en la sección 3.7.1 y siguientes y las adaptamos para recoger el caso que pretendemos. Fijamos $\mathcal{U}$ como universo algebraico para todo el capítulo.

4.2.1 Especificando implementaciones

En el capítulo anterior el primer paso fue definir, para cada G-tipo $\underline{\mathtt{T}}$, una Σ_{imp}-álgebra que denominamos $\mathcal{M}^{\underline{\mathtt{T}}}$ y que representa a una familia de Σ-álgebras definidas a partir de implementaciones de $\mathcal{T}$. Esta Σ_{imp}-álgebra posee una propiedad de coproducto y es final en una categoría de familias de Σ-álgebras definidas a partir de implementaciones. Como ya hemos comentado, la propiedad de coproducto se traduce en ser la mínima familia que contiene a todas las Σ-álgebras que se pueden definir a partir de una categoría concreta de implementaciones, pudiendo ver cada una de éstas incluida en ella. La propiedad de finalidad se traduce, en este caso, en una propiedad de generalidad, en el sentido de que el único paso natural desde cualquier otra familia de Σ-álgebras, dado por el morfismo final no es trivial sino que permite ver la familia de partida incluida en el objeto final.

Una de las presentaciones de cada objeto $\mathcal{M}^{\underline{\mathtt{T}}}$ se apoya en la descripción del conjunto de índices (el soporte asociado al género distinguido imp_Σ) como un espacio de funciones, siendo cada índice una tupla de funciones, una por cada operación de la signatura Σ, que provienen de una Σ-álgebra definida a partir de una implementación de $\mathcal{T}$. La importancia de esta Σ_{imp}-álgebra es que recoge la información mínima necesaria para tratar las álgebras como datos. El siguiente paso es el salto a la máquina implementando $\mathcal{M}^{\underline{\mathtt{T}}}$, y es éste el que permite justificar la construcción en tiempo de ejecución de representaciones en la máquina de Σ-álgebras.

Vamos a ocuparnos en primer lugar de definir un objeto que juegue el mismo papel que el objeto $\mathcal{M}^{\underline{\mathtt{T}}}$ en el capítulo anterior pero que recoja también información acerca de la represen-

tación sobre la que reposa cada implementación. En el capítulo anterior, a partir de un TAD $\mathcal{T}$ y su categoría de implementaciones $Imp(\mathcal{T})$ definimos la categoría $CImp(\mathcal{T})$ de Σ-álgebras definidas a partir de las implementaciones de $\mathcal{T}$, categoría a la que aplicamos la operación $()_{imp}$, de forma que los objetos de la categoría obtenida, $CImp(\mathcal{T}_{imp})$, representan familias de Σ-álgebras definidas a partir de implementaciones de $\mathcal{T}$. Fijando un G-tipo $\underline{\mathtt{T}}$ y considerando la subcategoría de álgebras de $CImp(\mathcal{T}_{imp})$ con G-tipo $\underline{\mathtt{T}}$ y morfismos identidad sobre $\underline{\mathtt{T}}$, el objeto $\mathcal{M}^{\underline{\mathtt{T}}}$ es final en ella. Para incluir los invariantes y las igualdades de las representaciones es necesario enriquecer la categoría $CImp(\mathcal{T}_{imp})$.

Comenzamos por la parte sintáctica de la operación. Sea Σ una signatura y Σ_{imp} la signatura obtenida aplicando a Σ la operación $()_{imp}$. Definimos una operación $()_{rep}$ que construirá a partir de una signatura Σ_{imp} la signatura que denotaremos por $\Sigma_{imp_{rep}}$ definida como sigue:

- $G_{imp_{rep}} = G_{imp} \cup \{bool\}$
- $\Omega_{imp_{rep}} = \Omega_{imp} \cup \{rep\_invar_g \colon imp_\Sigma\ g \to bool\}_{g \in G} \cup \{rep\_igual_g \colon imp_\Sigma\ g\ g \to bool\}_{g \in G}$

La intención al incluir el género *bool* es representar los valores booleanos. Si en Σ ya existe un género con esa intención, se renombra a *bool*, así, en algunos casos, el conjunto de géneros de Σ_{imp} y el de $\Sigma_{imp_{rep}}$ coinciden.

La signatura $\Sigma_{imp_{rep}}$ recoge la parte sintáctica de la especificación de familias de álgebras que provienen de implementaciones de $\mathcal{T}$ sobre representaciones con invariantes e igualdades de representación calculables y explícitos, ya que la inclusión en $\Omega_{imp_{rep}}$ de las operaciones rep_invar_g y rep_igual_g para cada $g \in G$, se hace con la intención de que recojan la parte de representación. Es por este motivo por el que empleamos el prefijo *rep* en las operaciones, así como en el nombre de la operación $()_{rep}$. Abusando de la notación, la interpretación de cada operación rep_invar_g en un álgebra A, $(rep\_invar_g)_A$, la denotaremos por $rep\_invar_{A_g}$ y, análogamente para las operaciones rep_igual_g.

Observar que la operación se aplica sobre signaturas resultado de la operación $()_{imp}$ y no a cualquier signatura ya que la operación $()_{rep}$ está asociada con las representaciones, no con las álgebras, luego lo natural es definir la operación sobre signaturas que recogen la parte sintáctica de las implementaciones y éstas son las obtenidas por aplicación de la operación $()_{imp}$.

Denotamos por $CImp_{rep}(\mathcal{T}_{imp})$ a la categoría de $\Sigma_{imp_{rep}}$-álgebras que juega un papel análogo al de la categoría $CImp(\mathcal{T}_{imp})$ en el capítulo anterior, es decir, sus objetos son $\Sigma_{imp_{rep}}$-álgebras que recogen información sobre familias de implementaciones de $\mathcal{T}$ con determinadas propiedades. La categoría $CImp_{rep}(\mathcal{T}_{imp})$ se define como sigue:

- Los objetos son las $\Sigma_{imp_{rep}}$-álgebras $A = \langle \mathtt{T}_{imp_\Sigma}, (\mathtt{T}_g)_{g \in G}, \mathtt{T}_{bool},\ \{imp\_\sigma_A \colon \mathtt{T}_{imp_\Sigma} \times \mathtt{T}_\omega \to \mathtt{T}_v,\ Def(imp\_\sigma_A)\}_{(\sigma \colon \omega \to v) \in \Omega},\ \{rep\_invar_{A_g} \colon \mathtt{T}_{imp_\Sigma} \times \mathtt{T}_g \to \mathtt{T}_{bool},\ Def(rep\_invar_{A_g})\}_{g \in G},\ \{rep\_igual_{A_g} \colon \mathtt{T}_{imp_\Sigma} \times \mathtt{T}_g \times \mathtt{T}_g \to \mathtt{T}_{bool},\ Def(rep\_igual_{A_g})\}_{g \in G} \rangle$ tales que, para cada $\mathtt{d} \in \mathtt{T}_{imp_\Sigma}$, existe $\mathcal{J} = (\underline{\mathcal{R}}, (\mathtt{p}^{\mathcal{J}}_\sigma)_{\sigma \in \Omega})$ implementación de $\mathcal{T}$ con G-tipo $\underline{\mathtt{T}}$, invariantes e igualdades de representación calculables y explícitos implementados por las familias de programas $(\mathtt{p}^{\mathcal{J}}_{invar_{\mathcal{R}_g}})_{g \in G}$ y $(\mathtt{p}^{\mathcal{J}}_{igual_{\mathcal{R}_g}})_{g \in G}$, verificando las funciones definidas por los programas las siguientes condiciones:
  - Para cada $g \in G$, la función definida por el programa $\mathtt{p}^{\mathcal{J}}_{invar_{\mathcal{R}_g}}$ coincide con la función $rep\_invar_{A_g}(\mathtt{d}, -)$; y, análogamente, la función definida por el programa $\mathtt{p}^{\mathcal{J}}_{igual_{\mathcal{R}_g}}$ es la función $rep\_igual_{A_g}(\mathtt{d}, -)$.

Los dominios de definición de las funciones $rep\_invar_{A_g}(\mathtt{d}, -)$ y $rep\_igual_{A_g}(\mathtt{d}, -)$, así como su comportamiento, están determinados por coincidir con las funciones definidas por los programas que implementan a los invariantes y a las igualdades de las representaciones. Así, para cada $g \in G$, la función $rep\_invar_{A_g}(\mathtt{d}, -)\colon \mathtt{T}_g \to \mathtt{T}_{bool}$ es una función total que además define el invariante de la representación $\mathcal{R}_g$, por lo que su comportamiento se describe del siguiente modo: $rep\_invar_{A_g}(\mathtt{d}, -)(\mathtt{d}_g) \neq \mathtt{nil}$ si y solo si $\mathtt{d}_g \in \mathcal{S}_g^{\mathcal{J}}$, donde $\mathcal{S}_g^{\mathcal{J}}$ denota al soporte de la representación $\mathcal{R}_g$. Análogamente podemos determinar el dominio de la función $rep\_igual_{A_g}(\mathtt{d}, -)\colon \mathtt{T}_g \times \mathtt{T}_g \to \mathtt{T}_{bool}$ como $\mathcal{S}_g^{\mathcal{J}} \times \mathcal{S}_g^{\mathcal{J}}$, así como su comportamiento, siendo $rep\_igual_{A_g}(\mathtt{d}, -)(\mathtt{d}_g, \mathtt{d}_g') \neq \mathtt{nil}$ si y solo si $\mathtt{d}_g =_{\mathcal{R}_g} \mathtt{d}_g'$, donde, como es natural, $=_{\mathcal{R}_g}$ denota la igualdad de la representación $\mathcal{R}_g$.

· Las funciones definidas por los programas $(\mathtt{p}_\sigma^{\mathcal{J}})_{\sigma\in\Omega}$ coinciden con las de la Σ-álgebra $A_{\mathtt{d}} = \left\langle (\mathtt{T}_g)_{g\in G},\ \{imp\_\sigma_A(\mathtt{d}, -)\colon \mathtt{T}_\omega \to \mathtt{T}_v,\ Def(imp\_\sigma_A(\mathtt{d}, -))\}_{(\sigma\colon \omega\to v)\in\Sigma}\right\rangle$ con dominios de definición dependientes de los soportes de las representaciones $(\mathcal{S}_g^{\mathcal{J}})_{g\in G}$ y totales sobre éstos. Así, para cada $(\sigma\colon \omega \to v) \in \Omega$, $Def(imp\_\sigma_A(\mathtt{d}, -)) = \mathcal{S}_\omega^{\mathcal{J}}$ donde $\mathcal{S}_\omega^{\mathcal{J}}$ denota al producto $\mathcal{S}_{g_1}^{\mathcal{J}} \times \cdots \times \mathcal{S}_{g_n}^{\mathcal{J}}$ con $\omega = g_1 \ldots g_n$. Como los invariantes aparecen explícitamente, lo natural es que sean ellos los que determinen el dominio de los programas de la implementación $\mathcal{J}$, y de ahí que se exija que las operaciones de cada Σ-álgebra $A_{\mathtt{d}}$ tengan como dominio los productos de los soportes de las representaciones.

- Si A y B son $\Sigma_{imp_{rep}}$-álgebras de $\mathcal{C}Imp_{rep}(\mathcal{T}_{imp})$, un $\Sigma_{rep_{imp}}$-morfismo $f\colon A \to B$, $f = (f_{imp_\Sigma}, (f_g)_{g\in G}, f_{bool})$, es un morfismo de $\mathcal{C}Imp_{rep}(\mathcal{T}_{imp})$ si, para todo $\mathtt{d} \in A_{imp_\Sigma}$, $(f_g)_{g\in G}\colon A_{\mathtt{d}} \to B_{f_{imp_\Sigma}(\mathtt{d})}$ es un morfismo entre las Σ-álgebras y $f_{bool}\colon \mathtt{T}_{A_{bool}} \to \mathtt{T}_{B_{bool}}$ verifica, para cada $g \in G$, las siguientes condiciones:

  · $f_g(\mathcal{S}_g^{\mathcal{J}^A}) \subseteq \mathcal{S}_g^{\mathcal{J}^B}$;
  · $f_{bool} \circ rep\_invar_{A_g}(\mathtt{d}, -) = rep\_invar_{B_g}(f_{imp_\Sigma}(\mathtt{d}), -) \circ f_g$;
  · $f_{bool} \circ rep\_igual_{A_g}(\mathtt{d}, -) = rep\_igual_{B_g}(f_{imp_\Sigma}(\mathtt{d}), -) \circ (f_g, f_g)$.

  Las condiciones anteriores se exigen para que haya conmutatividad no solo sobre los operadores de las Σ-álgebras sino también sobre los que definen los invariantes y las igualdades de las representaciones.

Es claro, por todo lo mostrado en los capítulos anteriores, que cada $\Sigma_{imp_{rep}}$-álgebra de $\mathcal{C}Imp_{rep}(\mathcal{T}_{imp})$ recoge una familia de implementaciones del TAD $\mathcal{T}$ sobre representaciones con invariantes e igualdades de representación calculables y explícitas, con funciones definidas por los programas totales sobre los soportes, todas ellas con el mismo G-tipo y recogiendo cada una de ellas una parte de representación y otra de implementación.

Sin pérdida de generalidad, para lo que sigue fijamos como soporte para el género *bool*, en cualquier $\Sigma_{imp_{rep}}$-álgebra, el universo $\mathcal{U}$ interpretando cualquier objeto distinto de `nil` (por la igualdad `eq`) como *true*.

Veamos un ejemplo de $\Sigma_{imp_{rep}}$-álgebra.

Ejemplo 4.2.1 Consideramos el TAD $\mathcal{T}_{\mathtt{GRP}}$ de los grupos cociente de $\mathbb{Z}$ módulo n, para $n > 1$. Consideramos la familia de implementaciones del TAD $\mathcal{T}_{\mathtt{GRP}}$, $\{\mathcal{I}_{\mathtt{GRP}}^n\}_{n\in\mathbb{N},n>1}$, una para cada $\mathbb{Z}/n\mathbb{Z}$ con $n \in \mathbb{N}$, $n > 1$, definida en la introducción de este capítulo (página 205). Todas ellas son

implementaciones del TAD $\mathcal{T}_{\texttt{GRP}}$ con G-tipo `integer`. Para obtener datos en la máquina que sean representación de alguno de los grupos anteriores, la primera opción pasa por considerar la signatura $\texttt{GRP}_{imp}$. Sin embargo, haciéndolo así no recogemos información acerca de las representaciones. Para recoger también esta información, consideramos la signatura $\texttt{GRP}_{imp_{rep}}$, signatura con géneros $imp_{\texttt{GRP}}$, g y *bool* y con las siguientes operaciones:

$$\begin{array}{rcl}
imp_prd & : & imp_{\texttt{GRP}}\ g\ g \to g \\
imp_inv & : & imp_{\texttt{GRP}}\ g \to g \\
imp_unt & : & imp_{\texttt{GRP}} \to g \\
rep_invar & : & imp_{\texttt{GRP}}\ g \to bool \\
rep_igual & : & imp_{\texttt{GRP}}\ g\ g \to bool
\end{array}$$

Como la signatura de partida `GRP` solo posee un género, no hay lugar a confusión si incluimos al aplicar la operación $()_{rep}$ las operaciones rep_invar y rep_igual sin hacer referencia al género al que están ligadas.

Definimos a continuación una $\texttt{GRP}_{imp_{rep}}$-álgebra A que recoge a las implementaciones de la familia $\{\mathcal{I}^n_{\texttt{GRP}}\}_{n\in\mathbb{N},n>1}$. Tomamos como conjunto soporte para el género g el tipo `integer` (y, como hemos comentado el universo $\mathcal{U}$ para *bool*). Para el género $imp_{\texttt{GRP}}$ tomamos como conjunto soporte el tipo `(satisfies positivoMayorQue1p)`, tipo que nos permite indexar todas las implementaciones consideradas, y que se define a partir del siguiente predicado

```
(defun positivoMayorQue1p( n )
            (and integerp (> n 1)))
```

Definimos las operaciones de la $\texttt{GRP}_{imp_{rep}}$-álgebra A como sigue:

- Como función rep_invar_A: (`satisfies positivoMayorQue1p`) $\times$ `integer` $\to \mathcal{U}$, la función total dada por $rep\_invar_A(\texttt{n},\texttt{d}) :=$ `t` si $\texttt{0} \leq \texttt{d} < \texttt{n}$, y $rep\_invar_A(\texttt{n},\texttt{d}) :=$ `nil` en otro caso. Para cada `n` $\in$ `(satisfies positivoMayorQue1p)`, el conjunto $\{\texttt{d} \in \texttt{integer} \mid rep\_invar_A(\texttt{n},\texttt{d}) = \texttt{t}\}$ es $\{0,\ldots,\texttt{n}-1\}$, conjunto que coincide con $\mathcal{S}^n_A$=`(mod n)`, soporte de la representación sobre la que está definida la implementación correspondiente a $\mathbb{Z}/n\mathbb{Z}$. Los dominios de definición del resto de operaciones se definen en función de los conjuntos $\mathcal{S}^n_A$.

- La función rep_igual_A: (`satisfies positivoMayorQue1p`) $\times$ `integer` $\times$ `integer` $\to \mathcal{U}$ tiene por dominio $Def(rep\_igual_A) = \{(\texttt{n},\texttt{d}_1,\texttt{d}_2) \mid \texttt{d}_1,\texttt{d}_2 \in \mathcal{S}^n_A\}$ y viene dada por $rep\_igual_A(\texttt{n},\texttt{d}_1,\texttt{d}_2) :=$ `(= d1 d2)`. (La igualdad (=) utilizada en la definición de la función anterior es la específica del tipo `integer`.)

- La función imp_prd_A: (`satisfies positivoMayorQue1p`) $\times$ `integer` $\times$ `integer` $\to$ `integer` tiene por dominio $Def(imp\_prd_A) = \{(\texttt{n},\texttt{d}_1,\texttt{d}_2) \mid \texttt{d}_1,\texttt{d}_2 \in \mathcal{S}^n_A\}$ y se define como $imp\_prd_A(\texttt{n},\texttt{d}_1,\texttt{d}_2)$:= `(mod (+ d1 d2) n)`.

- imp_inv_A: (`satisfies positivoMayorQue1p`) $\times$ `integer` $\to$ `integer` es la función con dominio $\{(\texttt{n},\texttt{d}) \mid \texttt{d} \in \mathcal{S}^n_A\}$ definida por $imp\_inv_A(\texttt{n},\texttt{d})$:= `(mod (- d) n)`; y, finalmente,

- imp_unt_A: (`satisfies positivoMayorQue1p`) $\to$ `integer` es la función total definida como $imp\_unt_A(\mathtt{n}) := \mathtt{0}$

La $\mathtt{GRP}_{imp_{rep}}$-álgebra anterior es un objeto de la categoría $CImp_{rep}(\mathcal{T}_{\mathtt{GRP}_{imp}})$ ya que, para cada $\mathtt{n} \in$ `(satisfies positivoMayorQue1p)`, la implementación $\mathcal{I}^n_{\mathtt{GRP}}$ definida al principio del capítulo, es una implementación del TAD $\mathcal{T}_{\mathtt{GRP}}$ con G-tipo `integer` y que tiene como funciones definidas por sus programas a las funciones $\{imp prd_A(\mathtt{n}, -), imp\_inv_A(\mathtt{n}, -), imp\_unt_A(\mathtt{n}, -)\}$. Además, para cada $\mathtt{n} \in$ `(satisfies positivoMayorQue1p)`, la representación sobre la que reposa la implementación $\mathcal{I}^n_{\mathtt{GRP}}$ posee invariante e igualdad de representación calculables. Para cada $\mathtt{n} \in$ `(satisfies positivoMayorQue1p)`, el siguiente programa implementa al invariante

$\mathtt{p}^{\mathtt{n}}_{invar} = (\mathtt{c}^{\mathtt{p}^{\mathtt{n}}_{invar}}, \mathtt{integer}, \mathtt{integer}, \mathcal{U})$ con

$$\mathtt{c}^{\mathtt{p}^{\mathtt{n}}_{invar}} \equiv \mathtt{\#'(lambda\ (d)\ (and\ (>=\ d\ 0)\ (<\ d\ n)))}$$

Análogamente, la igualdad de la representación es implementada por el siguiente programa

$\mathtt{p}^{\mathtt{n}}_{igual} = (\mathtt{c}^{\mathtt{p}^{\mathtt{n}}_{igual}}, \mathtt{(mod\ \ n)} \times \mathtt{(mod\ n)}, \mathtt{integer}, \mathtt{integer}, \mathcal{U})$

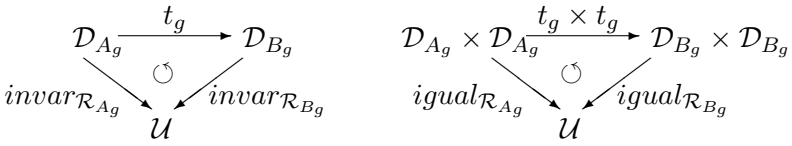
siendo $\mathtt{c}^{\mathtt{p}^{\mathtt{n}}_{invar}} \equiv$`#'=`.

□

Apoyándonos directamente en la categoría de implementaciones del TAD $\mathcal{T}$, podemos definir la categoría $CImp_{rep}(\mathcal{T}_{imp})$ de forma más compacta. Para ello, consideramos una subcategoría plena de la categoría de implementaciones de $\mathcal{T}$, categoría que denotaremos por $Imp_{Dec}(\mathcal{T})$. Los objetos de esta categoría son las implementaciones de $\mathcal{T}$ sobre representaciones con invariantes e igualdades de representación calculables y explícitos y con programas que implementan a las operaciones totales sobre los productos de los soportes de las representaciones. Así, cada implementación $\mathcal{J} = (\underline{\mathcal{R}}, (\mathtt{p}^{\mathcal{J}}_{\sigma})_{\sigma\in\Omega})$ de esta categoría tiene asociadas dos familias de programas, una de ellas formada por los programas que implementan a los invariantes y otra por los que implementan a las igualdades de las representaciones. Así, un objeto $\mathcal{J}$ de esta categoría podemos expresarlo del siguiente modo: $\mathcal{J} = \big(\underline{\mathcal{R}}, (\mathtt{p}^{\mathcal{J}}_{invar_{\mathcal{R}_g}})_{g\in G}, (\mathtt{p}^{\mathcal{J}}_{igual_{\mathcal{R}_g}})_{g\in G}, (\mathtt{p}^{\mathcal{J}}_{\sigma})_{\sigma\in\Omega}\big)$, siendo para cada $g \in G$, $\mathtt{p}^{\mathcal{J}}_{invar_{\mathcal{R}_g}}$ un programa que implementa al invariante de la representación $\mathcal{R}_g$ y $\mathtt{p}^{\mathcal{J}}_{igual_{\mathcal{R}_g}}$ uno que implementa a la igualdad de la representación $\mathcal{R}_g$. Observar también que, para cada $g \in G$, el invariante de cada representación $\mathcal{R}_g$ es una función total $invar_{\mathcal{R}_g}\colon \mathcal{D}_g \to \mathcal{U}$ que define el soporte, así un programa que lo implemente tendrá como tipo de entrada $\mathcal{D}_g$, siendo $\mathcal{D}_g$ el dominio de la representación $\mathcal{R}_g$, a $\mathcal{U}$ como tipo de salida, $\mathcal{D}_g$ como dominio de definición y código $\mathtt{c}^{\mathtt{p}^{\mathcal{J}}_{invar_{\mathcal{R}_g}}}$ verificando para cada $\mathtt{d} \in \mathcal{D}_g$ que $\mathtt{c}^{\mathtt{p}^{\mathcal{J}}_{invar_{\mathcal{R}_g}}}(\mathtt{d}) \neq_{\mathtt{eq}} \mathtt{nil}$ si y solo si $\mathtt{d} \in \mathcal{S}_g$. Análogamente, para cada $g \in G$, $igual_{\mathcal{R}_g}\colon \mathcal{D}_g \times \mathcal{D}_g \to \mathcal{U}$ es una función parcial con dominio $\mathcal{S}_g \times \mathcal{S}_g$ que define la igualdad de la representación, por lo que un programa que la implemente tendrá como tipos de entrada $\mathcal{D}_g$ y $\mathcal{D}_g$, $\mathcal{U}$ como tipo de salida, como dominio de definición $\mathcal{S}_g \times \mathcal{S}_g$ y código $\mathtt{c}^{\mathtt{p}^{\mathcal{J}}_{igual_{\mathcal{R}_g}}}$ verificando para cada $\mathtt{d}, \mathtt{d}' \in \mathcal{D}_g$ que $\mathtt{c}^{\mathtt{p}^{\mathcal{J}}_{igual_{\mathcal{R}_g}}}(\mathtt{d}, \mathtt{d}') \neq_{\mathtt{eq}} \mathtt{nil}$ si y solo si $\mathtt{d} =_{\mathcal{R}_g} \mathtt{d}'$. Además, para cada $(\sigma\colon \omega \to v) \in \Omega$, la función definida por el programa $\mathtt{p}^{\mathcal{J}}_{\sigma}$ tiene por dominio $\mathcal{S}_{\omega} = \mathcal{S}_{g_1} \times \cdots \times \mathcal{S}_{g_n}$ siendo $\omega = g_1 \ldots g_n$.

En este punto consideramos necesario un comentario sobre el hecho de tomar $Imp_{Dec}(\mathcal{T})$ como subcategoría plena de la categoría de implementaciones de $\mathcal{T}$. Hemos tomado $\mathcal{U}$ como tipo

de salida de los programas que implementan a los invariantes y a las igualdades, interpretando `nil` como el valor booleano *false* y cualquier otro objeto Lisp distinto de `nil` (por `eq`) como el valor *true*. A las i–transformaciones se les exige conmutatividad con las operaciones y, en este caso, siendo que estamos considerando las funciones $invar_{\mathcal{R}_g}$ e $igual_{\mathcal{R}_g}$ para cada $g \in G$, es natural exigirla también sobre ellas. Así si $(t, f)\colon \mathcal{J}_A \to \mathcal{J}_B$ es una i–transformación, los siguientes diagramas recogen las condiciones de conmutatividad sobre invariantes e igualdades.



Por definición de i–transformación se verifica que $t_g(\mathcal{S}_{A_g}) \subseteq \mathcal{S}_{B_g}$, por lo que los dominios de las funciones $invar_{\mathcal{R}_{B_g}} \circ t_g$ e $invar_{\mathcal{R}_{A_g}}$ coinciden. Lo anterior unido a que $invar_{\mathcal{R}_{A_g}}$ e $invar_{\mathcal{R}_{B_g}}$ definen los invariantes de las representaciones, hace que los diagramas correspondientes a los invariantes conmuten. Lo mismo ocurre con los diagramas correspondientes a las igualdades. Por ello, cualquier i–transformación entre implementaciones de $Imp_{Dec}(\mathcal{T})$ hace todos los diagramas anteriores conmutativos, lo que nos permite trabajar con una subcategoría plena de $Imp(\mathcal{T})$.

Usando esta categoría de implementaciones, la categoría $\mathcal{C}Imp_{rep}(\mathcal{T}_{imp})$ podemos verla como la subcategoría plena de $PAlg(\Sigma_{imp_{rep}})$ con objetos las $\Sigma_{imp_{rep}}$-álgebras $A = \langle \mathtt{T}_{imp_\Sigma}, (\mathtt{T}_g)_{g\in G}, \mathtt{T}_{bool}, \{imp\_\sigma_A\colon \mathtt{T}_{imp_\Sigma} \times \mathtt{T}_\omega \to \mathtt{T}_v,\ Def(imp\_\sigma_A)\}_{(\sigma\colon \omega\to v)\in\Omega}, \{rep\_invar_{A_g}\colon \mathtt{T}_{imp_\Sigma} \times \mathtt{T}_g \to \mathtt{T}_{bool}, Def(rep\_invar_{A_g})\}_{g\in G}, \{rep\_igual_{A_g}\colon \mathtt{T}_{imp_\Sigma} \times \mathtt{T}_g \times \mathtt{T}_g \to \mathtt{T}_{bool}, Def(rep\_igual_{A_g})\}_{g\in G} \rangle$ tales que, para cada Σ-álgebra $A_{\mathtt{d}} = \langle (\mathtt{T}_g)_{g\in G}, \{imp\_\sigma_A(\mathtt{d}, -)\colon \mathtt{T}_\omega \to \mathtt{T}_v,\ Def(imp\_\sigma_A(\mathtt{d}, -))\}_{(\sigma\colon \omega\to v)\in\Sigma} \rangle$ existe $\mathcal{J} = \big(\underline{\mathcal{R}},\ (\mathtt{p}^{\mathcal{J}}_{invar_{\mathcal{R}_g}})_{g\in G},\ (\mathtt{p}^{\mathcal{J}}_{igual_{\mathcal{R}_g}})_{g\in G},\ (\mathtt{p}^{\mathcal{J}}_{\sigma})_{\sigma\in\Omega}\big)$, implementación de la categoría $Imp_{Dec}(\mathcal{T})$, con G-tipo $\underline{\mathtt{T}}$, invariantes e igualdades de representación dadas por $rep\_invar_{A_g}(\mathtt{d}, -)$ y $rep\_igual_{A_g}(\mathtt{d}, -)$ respectivamente, para cada $g \in G$, y funciones definidas por los programas que implementan a las operaciones de la Σ-álgebra las funciones $imp\_\sigma_A(\mathtt{d}, -)$, para cada $\sigma \in \Sigma$. Implícitamente se tiene que, para cada $g \in G$, $F(\mathtt{p}^{\mathcal{J}}_{invar_{\mathcal{R}_g}}) = rep\_invar_{A_g}(\mathtt{d}, -)$ y $F(\mathtt{p}^{\mathcal{J}}_{igual_{\mathcal{R}_g}}) = rep\_igual_{A_g}(\mathtt{d}, -)$; y, para cada $\sigma \in \Sigma$, $F(\mathtt{p}^{\mathcal{J}}_{\sigma}) = imp\_\sigma_A(\mathtt{d}, -)$.

Para continuar el estudio del mismo modo que en el capítulo anterior, nuestro primer objetivo es demostrar la existencia de una familia de objetos de la categoría $\mathcal{C}Imp_{rep}(\mathcal{T}_{imp})$ que cubra a todos los objetos de la categoría, es decir, buscamos objetos con propiedades de coproducto en $\mathcal{C}Imp_{rep}(\mathcal{T}_{imp})$. La idea es que cada uno de estos objetos recoja a todos los de la categoría con determinado G-tipo. Si además, estos objetos poseen propiedades de finalidad, de forma que los morfismos finales puedan verse como inclusiones, de alguna manera tenemos la generalidad de estos objetos.

Así, descomponemos la categoría $\mathcal{C}Imp_{rep}(\mathcal{T}_{imp})$ en subcategorías con conjuntos soporte fijos para los géneros de Σ (teniendo en cuenta que el conjunto soporte para *bool* está fijo y es $\mathcal{U}$ y que el de imp_Σ queda libre), con el fin de agrupar aquellas implementaciones que trabajan sobre los mismos datos. Fijamos un G-tipo $\underline{\mathtt{T}} = (\mathtt{T}_g)_{g\in G}$ donde para cada $g \in G$, $\mathtt{T}_g$ es un tipo concreto Common Lisp. Como ya hemos visto en los capítulos anteriores, el marco de trabajo considera morfismos que son la identidad sobre los géneros de la signatura de partida, en este caso Σ, por lo que al definir las categorías que nos interesan, restringimos los morfismos. Denotamos por $\mathcal{C}Imp^{\underline{\mathtt{T}},\{\}}_{rep}(\mathcal{T}_{imp})$ a la subcategoría de $\mathcal{C}Imp_{rep}(\mathcal{T}_{imp})$ con objetos aquellos que poseen a $\underline{\mathtt{T}}$ como conjuntos soporte para los géneros de Σ y a $\mathcal{U}$ para *bool* (quedando libre el conjunto soporte

para el género imp_Σ), y con morfismos los de $\mathcal{C}Imp_{rep}(\mathcal{T}_{imp})$ que son la identidad sobre los géneros de Σ y *bool*.

Para simplificar la notación, definimos el siguiente conjunto

$$\Delta_\Sigma = \{\sigma\}_{\sigma\in\Omega} \cup \{invar_g\}_{g\in G} \cup \{igual_g\}_{g\in G}$$

que utilizaremos como conjunto de índices. Con esta notación, de ahora en adelante, representaremos cada objeto de la categoría $Imp_{Dec}(\mathcal{T})$ como $\mathcal{I} = (\underline{\mathcal{R}}, (\mathtt{p}_\delta)_{\delta\in\Delta_\Sigma})$.

Para cada G-tipo $\underline{\mathtt{T}}$, la categoría $\mathcal{C}Imp_{rep}^{\underline{\mathtt{T}},\{\}}(\mathcal{T}_{imp})$ posee un objeto con buenas propiedades desde el punto de vista de la Teoría de Categorías. Denotamos a este objeto por $\mathcal{M}^{rep,\underline{\mathtt{T}}}$ y una de sus descripciones es la siguiente:

- para cada $g \in G$, $\mathcal{M}_g^{rep,\underline{\mathtt{T}}} = \mathtt{T}_g$ y $\mathcal{M}_{bool}^{rep,\underline{\mathtt{T}}} = \mathcal{U}$;
- como conjunto soporte para el género distinguido, $\mathcal{M}_{imp_\Sigma}^{rep,\underline{\mathtt{T}}}$, tomamos

$$\mathcal{M}_{imp_\Sigma}^{rep,\underline{\mathtt{T}}} = \Big\{\overline{f} = (F(\mathtt{p}_\delta))_{\delta\in\Delta_\Sigma} \;\mid\; \exists\, \mathcal{I} = (\underline{\mathcal{R}}, (\mathtt{p}_\delta)_{\delta\in\Delta_\Sigma}) \in Obj(Imp_{Dec}(\mathcal{T})) \text{ tal que } \underline{\mathtt{T}} \text{ es el } G\text{-tipo de } \underline{\mathcal{R}}\Big\}$$

  Cada elemento del conjunto $\mathcal{M}_{imp_\Sigma}^{rep,\underline{\mathtt{T}}}$ es una tupla de funciones $\overline{f} = (f_\delta)_{\delta\in\Delta_\Sigma}$ indexada por el conjunto de índices Δ_Σ y que proviene de una implementación de la categoría $Imp_{Dec}(\mathcal{T})$;

- cada operación de $\mathcal{M}^{rep,\underline{\mathtt{T}}}$ se define como la proyección de la tupla de funciones en la componente correspondiente. Formalmente:

  · para cada $(\sigma\colon \omega \to v) \in \Omega$, la operación $imp\_\sigma_{\mathcal{M}^{rep,\underline{\mathtt{T}}}}\colon \mathcal{M}_{imp_\Sigma}^{rep,\underline{\mathtt{T}}} \times \mathtt{T}_\omega \to \mathtt{T}_v$ se define de manera natural por $imp\_\sigma_{\mathcal{M}^{rep,\underline{\mathtt{T}}}}((f_\delta)_{\delta\in\Delta_\Sigma}, \mathtt{d}_\omega) := f_\sigma(\mathtt{d}_\omega)$, siendo su dominio de definición

$$Def(imp\_\sigma_{\mathcal{M}^{rep,\underline{\mathtt{T}}}}) = \Big\{\big((f_\delta)_{\delta\in\Delta_\Sigma}, \mathtt{d}_\omega\big) \in \mathcal{M}_{imp_\Sigma}^{rep,\underline{\mathtt{T}}} \times \mathtt{T}_\omega \;\mid\; \mathtt{d}_\omega \in Def(f_\sigma)\Big\}$$

  · para cada $g \in G$, la función total $rep\_invar_{\mathcal{M}_g^{rep,\underline{\mathtt{T}}}}\colon \mathtt{T}_{imp_\Sigma} \times \mathtt{T}_g \to \mathcal{U}$ viene dada por $rep\_invar_{\mathcal{M}_g^{rep,\underline{\mathtt{T}}}}((f_\delta)_{\delta\in\Delta_\Sigma}, \mathtt{d}) := f_{invar_g}(\mathtt{d})$;

  · para cada $g \in G$, la operación $rep\_igual_{\mathcal{M}_g^{rep,\underline{\mathtt{T}}}}\colon \mathtt{T}_{imp_\Sigma} \times \mathtt{T}_g \times \mathtt{T}_g \to \mathcal{U}$ con dominio de definición

$$Def(rep\_igual_{\mathcal{M}^{rep,\underline{\mathtt{T}}}}) = \Big\{\big((f_\delta)_{\delta\in\Delta_\Sigma}, \mathtt{d}_1, \mathtt{d}_2\big) \in \mathcal{M}_{imp_\Sigma}^{rep,\underline{\mathtt{T}}} \times \mathtt{T}_g \times \mathtt{T}_g \;\mid\; (\mathtt{d}_1, \mathtt{d}_2) \in Def(f_{igual_g})\Big\}$$

  y está definida por $rep\_igual_{\mathcal{M}_g^{rep,\underline{\mathtt{T}}}}\big((f_\delta)_{\delta\in\Delta_\Sigma}, \mathtt{d}_1, \mathtt{d}_2\big) := f_{igual_g}(\mathtt{d}_1, \mathtt{d}_2)$.

Cada $\overline{f} \in \mathcal{M}^{rep,\underline{\mathtt{T}}}_{imp_\Sigma}$ es una tupla de funciones que provienen de programas de una implementación de $Imp_{Dec}(\mathcal{T})$ por lo que sus dominios son conocidos a priori. Así, si $\overline{f} \in \mathcal{M}^{rep,\underline{\mathtt{T}}}_{imp_\Sigma}$, existe una implementación de $Imp_{Dec}(\mathcal{T})$, $\mathcal{I} = (\underline{\mathcal{R}}, (\mathtt{p}^{\mathcal{I}}_\delta)_{\delta\in\Delta_\Sigma})$ tal que, para cada $\delta \in \Delta_\Sigma$, la función f_δ es la definida por el programa $\mathtt{p}^{\mathcal{I}}_\delta$, y, por tanto, los dominios de definición de las funciones de $\overline{f}$ están determinados y son los siguientes:

- para cada $g \in G$, el invariante de la representación $\mathcal{R}_g$ es total, por tanto, $Def(f_{invar_g})$ coincide con $Def(F(\mathtt{p}^{\mathcal{I}}_{invar_g}))$ y es $\mathtt{T}_g$;
- para cada $g \in G$, la igualdad de la representación $\mathcal{R}_g$ es una relación definida sobre el soporte $\mathcal{S}_g$, por lo que $Def(f_{igual_g})$ debe ser $Def(F(\mathtt{p}^{\mathcal{I}}_{igual_g}))$ que es $\mathcal{S}_g \times \mathcal{S}_g$;
- para cada $(\sigma\colon \omega \to v) \in \Omega$, $Def(f_\sigma) = Def(F(\mathtt{p}^{\mathcal{I}}_\sigma)) = \mathcal{S}_\omega$, siendo $\mathcal{S}_\omega = \mathcal{S}_{g_1} \times \cdots \times \mathcal{S}_{g_n}$ con $\omega = g_1 \ldots g_n$.

Como puede verse en las expresiones anteriores, si $\overline{f} \in \mathcal{M}^{rep,\underline{\mathtt{T}}}_{imp_\Sigma}$ e $\mathcal{I} = (\underline{\mathcal{R}}, (\mathtt{p}^{\mathcal{I}}_\delta)_{\delta\in\Delta_\Sigma})$ es una implementación de $Imp_{Dec}(\mathcal{T})$ con G-tipo $\underline{\mathtt{T}}$ que hace que $\overline{f}$ pertenezca a $\mathcal{M}^{rep,\underline{\mathtt{T}}}_{imp_\Sigma}$, entonces para cada género g de Σ, el conjunto $\{\mathtt{d} \in \mathtt{T}_g \mid f_{invar_g}(\mathtt{d}) \neq_{\mathtt{eq}} \mathtt{nil}\}$ coincide con $\mathcal{S}_g$, soporte de la representación $\mathcal{R}_g$.

La $\Sigma_{imp_{rep}}$-álgebra $\mathcal{M}^{rep,\underline{\mathtt{T}}}$ definida es un objeto de $\mathcal{C}Imp^{\underline{\mathtt{T}},\{\}}_{rep}(\mathcal{T}_{imp})$ puesto que, cada $\overline{f} \in \mathcal{M}^{rep,\underline{\mathtt{T}}}_{imp_\Sigma}$ junto con el G-tipo $\underline{\mathtt{T}}$ y $\mathcal{U}$, definen una Σ-álgebra para la que existe un objeto de $Imp_{Dec}(\mathcal{T})$ que hace que $\overline{f}$ esté en $\mathcal{M}^{rep,\underline{\mathtt{T}}}_{imp_\Sigma}$, sirviendo esta misma implementación para garantizarlo.

El conjunto $\mathcal{M}^{rep,\underline{\mathtt{T}}}_{imp_\Sigma}$ recoge a todas las tuplas de funciones definidas por los programas de las implementaciones de $Imp_{Dec}(\mathcal{T})$, lo que hace que sea un objeto especial en la categoría $\mathcal{C}Imp^{\underline{\mathtt{T}}}_{rep}(\mathcal{T}_{imp})$ cuyas propiedades estudiamos a continuación.

Denotamos por $Imp^{\underline{\mathtt{T}}}_{Dec}(\mathcal{T})$ a la subcategoría plena de $Imp_{Dec}(\mathcal{T})$ con objetos las implementaciones con G-tipo $\underline{\mathtt{T}}$. Cada objeto de la categoría $\mathcal{C}Imp^{\underline{\mathtt{T}}}_{rep}(\mathcal{T}_{imp})$ representa una familia de implementaciones de $Imp_{Dec}(\mathcal{T})$. Esto hace que podamos ver cada implementación de $Imp^{\underline{\mathtt{T}}}_{Dec}(\mathcal{T})$ como un objeto de la categoría $\mathcal{C}Imp^{\underline{\mathtt{T}}}_{rep}(\mathcal{T}_{imp})$, viéndola como una familia formada por un único objeto. Lo anterior queda formalmente explicado por la existencia de una aplicación $F\colon Obj(Imp^{\underline{\mathtt{T}}}_{Dec}(\mathcal{T})) \to Obj(\mathcal{C}Imp^{\underline{\mathtt{T}}}_{rep}(\mathcal{T}_{imp}))$ que lleva cada implementación de $Imp^{\underline{\mathtt{T}}}_{Dec}(\mathcal{T})$, $\mathcal{I} = (\underline{\mathcal{R}}, (\mathtt{p}_\delta)_{\delta\in\Delta_\Sigma})$, a la $\Sigma_{imp_{rep}}$-álgebra

$$F(\mathcal{I}) := \Big\langle \mathtt{null},\quad \underline{\mathtt{T}},\quad \mathcal{U},\quad \{imp\_\sigma_{F(\mathcal{I})},\quad Def(imp\_\sigma_{F(\mathcal{I})})\}_{\sigma\in\Omega},\quad \{rep\_invar_{F(\mathcal{I})_g},\ Def(rep\_invar_{F(\mathcal{I})_g})\}_{g\in G},\ \{rep\_igual_{F(\mathcal{I})_g},\ Def(rep\_igual_{F(\mathcal{I})_g})\}_{g\in G}\Big\rangle$$

(`null` es el tipo Common Lisp que tiene por único objeto a `nil`.) Para cada $g \in G$, $rep\_invar_{F(\mathcal{I})_g}$ es total y definida, para cada $\mathtt{d} \in \mathtt{T}_g$, por $rep\_invar_{F(\mathcal{I})_g}(\mathtt{nil}, \mathtt{d}) := F(\mathtt{p}^{\mathcal{I}}_{invar_g})(\mathtt{d})$; el dominio de $rep\_igual_{F(\mathcal{I})_g}$ es $\mathtt{null} \times Def(F(\mathtt{p}^{\mathcal{I}}_{igual_g}))$ y la función se define como $rep\_igual_{F(A)_g}(\mathtt{nil}, \mathtt{d}_1, \mathtt{d}_2) := F(\mathtt{p}^{\mathcal{I}}_{igual_g})(\mathtt{d}_1, \mathtt{d}_2)$, para todo $\mathtt{d}_1, \mathtt{d}_2 \in Def(F(\mathtt{p}^{\mathcal{I}}_{igual_g}))$. Para cada $\sigma \in \Omega$, el dominio de $imp\_\sigma_{F(\mathcal{I})}$ es $\mathtt{null} \times Def(F(\mathtt{p}^{\mathcal{I}}_\sigma))$ y su comportamiento viene descrito por $imp\_\sigma_{F(\mathcal{I})}(\mathtt{nil}, \mathtt{d}_\omega) := F(\mathtt{p}^{\mathcal{I}}_\sigma)(\mathtt{d}_\omega)$, para todo $\mathtt{d}_\omega \in Def(F(\mathtt{p}^{\mathcal{I}}_\sigma))$. Observar que la aplicación

F no es inyectiva ya que pueden existir distintas implementaciones del TAD $\mathcal{T}$ que definan el mismo objeto de $CImp_{rep}(\mathcal{T}_{imp})$. Pese a ello, la aplicación F puede verse como el paso evidente desde una implementación de $\mathcal{T}$ a una familia de implementaciones de $\mathcal{T}$ (la familia con un único objeto). Denotando por $Imp^{\underline{\mathtt{T}},\{\}}_{Dec}(\mathcal{T})$ a la subcategoría de $Imp^{\underline{\mathtt{T}}}_{Dec}(\mathcal{T})$ con los mismos objetos y morfismos únicamente las identidades, podemos extender la aplicación F a un funtor $F\colon Imp^{\underline{\mathtt{T}},\{\}}_{Dec}(\mathcal{T}) \to \mathcal{C}Imp^{\underline{\mathtt{T}},\{\}}_{rep}(\mathcal{T}_{imp})$ y obtener la propiedad del objeto $\mathcal{M}^{rep,\underline{\mathtt{T}}}$ recogida en el siguiente teorema.

Teorema 4.2.2 *El objeto $\mathcal{M}^{rep,\underline{\mathtt{T}}}$ es el coproducto en $\mathcal{C}Imp^{\underline{\mathtt{T}},\{\}}_{rep}(\mathcal{T}_{imp})$ de los objetos de la imagen de la inclusión de $Imp^{\underline{\mathtt{T}},\{\}}_{Dec}(\mathcal{T})$ en $\mathcal{C}Imp^{\underline{\mathtt{T}},\{\}}_{rep}(\mathcal{T}_{imp})$.*

El resultado anterior permite ver a $\mathcal{M}^{rep,\underline{\mathtt{T}}}$ como el objeto "más pequeño" de la categoría $\mathcal{C}Imp^{\underline{\mathtt{T}},\{\}}_{rep}(\mathcal{T}_{imp})$ que recoge a todas las implementaciones de la categoría $Imp^{\underline{\mathtt{T}},\{\}}_{Dec}(\mathcal{T})$.

El siguiente teorema recoge otra propiedad de tipo categorial del objeto $\mathcal{M}^{rep,\underline{\mathtt{T}}}$.

Teorema 4.2.3 *La $\Sigma_{imp_{rep}}$-álgebra $\mathcal{M}^{rep,\underline{\mathtt{T}}}$ es final en la categoría $\mathcal{C}Imp^{\underline{\mathtt{T}},\{\}}_{rep}(\mathcal{T}_{imp})$.*

La importancia de esta propiedad no radica tanto en la existencia de un paso natural desde cualquier otro objeto de la categoría, sino en que este paso no es trivial, permite incluir en el objeto final el resto de objetos.

Si denotamos por $\mathcal{C}Imp^{\{\}}_{rep}(\mathcal{T}_{imp})$ a la subcategoría de $\mathcal{C}Imp_{rep}(\mathcal{T}_{imp})$ con los mismos objetos y morfismos aquellos que son la identidad sobre los conjuntos soporte para los géneros de G y sobre $\mathcal{U}$, resulta que la familia de categorías $\left\{\mathcal{C}Imp^{\underline{\mathtt{T}},\{\}}_{rep}(\mathcal{T}_{imp})\right\}_{\underline{\mathtt{T}} \text{ es } G-\text{tipo}}$ define una partición de la categoría $\mathcal{C}Imp^{\{\}}_{rep}(\mathcal{T}_{imp})$, lo que, unido a la propiedad recogida en el anterior teorema verificada por cada objeto $\mathcal{M}^{rep,\underline{\mathtt{T}}}$ en su correspondiente categoría $\mathcal{C}Imp^{\underline{\mathtt{T}},\{\}}_{rep}(\mathcal{T}_{imp})$, permite obtener el siguiente resultado.

Teorema 4.2.4 *La familia $\{\mathcal{M}^{rep,\underline{\mathtt{T}}}\}_{\underline{\mathtt{T}} \text{ es } G-\text{tipo}}$ es final en la categoría $\mathcal{C}Imp^{\{\}}_{rep}(\mathcal{T}_{imp})$.*

Así, el objeto $\mathcal{M}^{rep,\underline{\mathtt{T}}}$ representa a la familia formada por todas las implementaciones de $\mathcal{T}$ con G-tipo $\underline{\mathtt{T}}$, invariantes e igualdades de representación calculables y programas totales sobre el producto de los soportes de las representaciones sobre las que están construidas. Además, concretamente, de cada implementación lo que queda recogido es una tupla de funciones formada por las definidas a partir de los programas que implementan a las operaciones de Ω y a los invariantes e igualdades de las representaciones. De esta forma, la representación en la máquina de una implementación estará formada por dos partes, una relativa a la representación y otra relativa a las operaciones. Además, es suficiente con ocuparnos de la implementación de estos objetos finales ya que, por un lado, en ellos quedan recogidas todas las implementaciones de $\mathcal{T}$ que interesan, y, por otro, desde cualquier otro objeto hay un paso natural y no trivial a uno de éstos.

4.2.2 Implementaciones canónicas

En esta sección nos ocupamos de implementar cada $\Sigma_{imp_{rep}}$-álgebra $\mathcal{M}^{rep,\underline{\mathtt{T}}}$. Los datos de $\mathcal{M}^{rep,\underline{\mathtt{T}}}$ son representaciones en la máquina de las implementaciones de $Imp^{\underline{\mathtt{T}}}_{Dec}(\mathcal{T})$, y, como se ha visto en el capítulo anterior, la manera más natural y general de implementar álgebras

definidas a partir de implementaciones es considerando como representación en la máquina de una implementación los objetos funcionales que forman parte de los programas. A este tipo de implementaciones es a las que hemos denominado implementaciones canónicas por ser las más generales, en el sentido de tener asegurada la existencia de un paso canónico desde cualquier otra implementación. En ellas es esencial el uso de la programación funcional. Sin embargo, pese a ser el modo más general, no es el único modo de implementar las álgebras $\mathcal{M}^{rep,\underline{\mathtt{T}}}$, tal y como hemos visto en el capítulo anterior a través de un ejemplo. Lo mismo ocurre en el caso que nos ocupa en el que la representación en la máquina de un álgebra está enriquecida considerando otros objetos funcionales adicionales respecto a la que teníamos, concretamente los correspondientes a invariantes e igualdades. Pese a ello, el tratamiento sigue siendo el mismo.

Comenzamos dando una implementación de un ejemplo de $\Sigma_{imp_{rep}}$-álgebra $\mathcal{M}^{rep,\underline{\mathtt{T}}}$ en la que el uso de la programación funcional no es esencial.

Ejemplo 4.2.5 Continuamos con el ejemplo del TAD $\mathcal{T}_{\mathtt{GRP}}$, TAD formado por los grupos cociente $\mathbb{Z}/n\mathbb{Z}$ con $n \in \mathbb{N}$, $n > 1$. Fijamos `integer` como conjunto soporte para el género g que representa a los elementos de los grupos y $\mathcal{U}$ para el género *bool*. Consideramos la $\mathtt{GRP}_{imp_{rep}}$-álgebra $\mathcal{M}^{\mathtt{integer},\mathcal{U}}$ que recoge a las implementaciones de los grupos $\mathbb{Z}/n\mathbb{Z}$ con $n \in \mathbb{N}$, $n > 1$, sobre representaciones con conjuntos soporte los fijados, invariantes e igualdades de representación calculables y explícitos y programas totales sobre los soportes de las representaciones. A continuación definimos una implementación del álgebra anterior:

- para el género de los elementos de los grupos tomamos la representación literal sobre `integer`;
- como representación para el género *bool*, la que interpreta cualquier objeto distinto de `nil` (por `eq`) como el valor booleano *true*, (representación introducida en el ejemplo 3.4.4);
- para el género distinguido $imp_{\mathtt{GRP}}$, tomamos la representación con dominio `integer`; soporte $\mathcal{S}_{imp_{\mathtt{GRP}}}$ formado por los objetos pertenecientes a `(satisfies positivoMayorQue1p)` (siendo `positivoMayorQue1p` el predicado definido en la página 211); función de abstracción $\alpha_{imp_{\mathtt{GRP}}}$: `integer` $\to \mathcal{M}^{rep,\mathtt{integer},\mathcal{U}}_{imp_{\mathtt{GRP}}}$ definida, para cada `n` del soporte $\mathcal{S}_{imp_{\mathtt{GRP}}}$ por: $\alpha_{imp_{\mathtt{GRP}}}(\mathtt{n}) := (f^{\mathtt{n}}_{invar}, f^{\mathtt{n}}_{igual}, f^{\mathtt{n}}_{prd}, f^{\mathtt{n}}_{inv}, f^{\mathtt{n}}_{unt})$, donde las funciones que forman la tupla anterior vienen dadas como sigue:
  - la función $f^{\mathtt{n}}_{invar}$: `integer` $\to \mathcal{U}$ es total y viene definida por $f^{\mathtt{n}}_{invar}(\mathtt{x}) := \mathtt{t}$ si `x` $\in <$ `n` $>$, siendo $<$ `n` $>$ el conjunto $\{0, \ldots, \mathtt{n}-1\}$; $f^{\mathtt{n}}_{invar}(\mathtt{x}) := \mathtt{nil}$, en caso contrario;
  - $f^{\mathtt{n}}_{igual}$: `integer` $\times$ `integer` $\to \mathcal{U}$ es la función parcial con dominio $<$ `n` $> \times <$ `n` $>$ y definida por $f^{\mathtt{n}}_{igual}(\mathtt{x},\mathtt{y})$:=`(= x y)`;
  - la función parcial $f^{\mathtt{n}}_{prd}$: `integer` $\times$ `integer` $\to$ `integer` tiene a $<$ `n` $> \times <$ `n` $>$ como dominio y viene definida por $f^{\mathtt{n}}_{prd}(\mathtt{x},\mathtt{y}) :=$ `(mod (+ x y) n)`;
  - la función $f^{\mathtt{n}}_{inv}$: `integer` $\to$ `integer` es también parcial, definida sobre el conjunto $<$ `n` $>$, y dada por $f^{\mathtt{n}}_{inv}(\mathtt{x}) :=$ `(mod (- x) n)`; finalmente,
  - la constante $f^{\mathtt{n}}_{unt}$ es la constante `0` $\in$ `integer`.

  Para cada `n` del soporte de la representación, la tupla de funciones $\alpha_{imp_{\mathtt{GRP}}}(\mathtt{n})$ está en el conjunto $\mathcal{M}^{rep,\mathtt{integer},\mathcal{U}}_{imp_{\mathtt{GRP}}}$, ya que añadiendo a la implementación $\mathcal{I}^{n}_{\mathtt{GRP}}$ del grupo $\mathbb{Z}/n\mathbb{Z}$ (definida en la página 205), los programas que implementan a los invariantes y a las igualdades

de las representaciones definidos en el ejemplo 4.2.1, se tiene una implementación del TAD $\mathcal{T}_{\texttt{GRP}}$ que pertenece a la categoría $Imp_{Dec}^{\texttt{integer}}(\mathcal{T}_{\texttt{GRP}})$ que lo garantiza.

- como programas que implementan a las operaciones consideramos los siguientes:

  · $\texttt{p}_{rep\_invar} = (\texttt{c}^{\texttt{p}_{rep\_invar}}, \mathcal{S}_{imp_{\texttt{GRP}}} \times \texttt{integer}, \texttt{integer}, \texttt{integer}, \mathcal{U})$ siendo su código el siguiente objeto funcional

  $$\texttt{c}^{\texttt{p}_{rep\_invar}} \equiv \texttt{\#'(lambda (n x) (and (<= 0 x) (< x n)))}$$

  · $\texttt{p}_{rep\_igual} = (\texttt{c}^{\texttt{p}_{rep\_igual}}, \mathcal{S}_{imp_{\texttt{GRP}}} \times < \texttt{n} > \times < \texttt{n} >, \texttt{integer}, \texttt{integer}, \texttt{integer}, \mathcal{U})$ con el siguiente objeto funcional como código

  $$\texttt{c}^{\texttt{p}_{rep\_igual}} \equiv \texttt{\#'(lambda (n x y) (= (mod x n) (mod y n)))}$$

  · $\texttt{p}_{imp\_prd} = (\texttt{c}^{\texttt{p}_{imp\_prd}}, \mathcal{S}_{imp_{\texttt{GRP}}} \times < \texttt{n} > \times < \texttt{n} >, \texttt{integer}, \texttt{integer}, \texttt{integer}, \texttt{integer})$ siendo su código el siguiente objeto funcional

  $$\texttt{c}^{\texttt{p}_{imp\_prd}} \equiv \texttt{\#'(lambda (n x y) (mod (+ x y) n))}$$

  · $\texttt{p}_{imp\_inv} = (\texttt{c}^{\texttt{p}_{imp\_inv}}, \mathcal{S}_{imp_{\texttt{GRP}}} \times < \texttt{n} >, \texttt{integer}, \texttt{integer}, \texttt{integer})$ siendo su código

  $$\texttt{c}^{\texttt{p}_{imp\_inv}} \equiv \texttt{\#'(lambda (n x) (mod (- x) n))}$$

  · $\texttt{p}_{imp\_unt} = (\texttt{c}^{\texttt{p}_{imp\_unt}}, \mathcal{S}_{imp_{\texttt{GRP}}}, \texttt{integer}, \texttt{integer})$ con código el siguiente objeto funcional

  $$\texttt{c}^{\texttt{p}_{imp\_unt}} \equiv \texttt{\#'(lambda (n) 0)}$$

En la implementación anterior no se usa de forma esencial la programación funcional, en el sentido de que las funciones no se utilizan como datos de primer orden, pese a que la descripción de la $\texttt{GRP}_{imp_{rep}}$-álgebra que se está implementando sea totalmente funcional.

□

La implementación anterior recoge a la familia de implementaciones $\{\mathcal{I}^n_{\texttt{GRP}}\}_{n\in\mathbb{N},n>1}$, dada al principio del capítulo, siendo cada $\mathcal{I}^n_{\texttt{GRP}}$ implementación del grupo $\mathbb{Z}/n\mathbb{Z}$ para cada $n \in \mathbb{N}$ con $n > 1$ (considerando como programas que implementan a invariantes e igualdades los dados en el ejemplo 4.2.1). En el marco del capítulo anterior no ha sido posible recoger en un álgebra la familia anterior puesto que los dominios de definición de los programas no factorizaban adecuadamente. En este sentido, vemos que este marco, al recoger como parte de la representación en la máquina de una implementación información de la representación sobre la que se ha construido, amplía el marco de formalización presentado en el capítulo anterior.

El ejemplo también muestra que no es necesario el uso de la programación funcional para implementar modelos funcionales. Sin embargo, como pasamos a ver a continuación, sí que el uso de ésta permite obtener implementaciones más generales y con propiedades deseables desde el punto de vista de la Teoría de Categorías.

Siguiendo el mismo esquema del capítulo anterior, introducimos a continuación una implementación concreta de $\mathcal{M}^{rep,\underline{\texttt{T}}}$, implementación basada en el uso de la programación funcional e inspirada en las implementaciones que hemos denominado canónicas en el capítulo anterior. Esta implementación recoge la parte de representación de las implementaciones, parte que no había sido incluida en el marco establecido en el capítulo anterior. La idea es almacenar una

implementación como una tupla de objetos funcionales, formada en este caso por los correspondientes a los programas que implementan a las operaciones de Ω y los que implementan a los invariantes y a las igualdades de las representaciones. El G-tipo fijado proporciona el patrón sintáctico de los datos sobre los que se aplicarán los objetos funcionales y algunos de estos objetos funcionales, concretamente los asociados a los invariantes, determinan los dominios de definición del resto de objetos funcionales de cada tupla.

Apoyándonos en que las signaturas con las que trabajamos son finitas, habíamos fijado una enumeración de las operaciones de Σ: $(\sigma_1, \ldots, \sigma_m)$. Como ahora trabajamos sobre $\Sigma_{imp_{rep}}$ necesitamos fijar una enumeración de sus operaciones, o lo que es lo mismo, apoyándonos en la enumeración anterior, consideramos una enumeración de los géneros de Σ: $(g_1, \ldots, g_n)$, enumeración que se traslada a las operaciones: $(rep\_invar_{g_1}, \ldots, rep\_invar_{g_n})$ y $(rep\_igual_{g_1}, \ldots, rep\_igual_{g_n})$. Estas enumeraciones las usaremos únicamente cuando sea necesario, en otro caso, seguiremos con la notación empleada hasta ahora.

Fijado el G-tipo $\underline{\mathtt{T}}$ y tomando $\mathcal{U}$ como tipo para *bool*, denotamos por $\mathcal{I}^{rep,\underline{\mathtt{T}},can}$ a la implementación de $\mathcal{M}^{rep,\underline{\mathtt{T}}}$ definida como sigue:

- Para cada $g \in G$, la representación sobre la que se define la implementación es la representación literal de $\mathtt{T}_g$; para el género *bool* se considera también la representación literal sobre $\mathcal{U}$ con la igualdad que interpreta como *true* cualquier objeto Lisp que no sea igual por `eq` al objeto `nil` (así es como hemos fijado el soporte para *bool*);

- Para el espacio funcional $\mathcal{M}^{rep,\underline{\mathtt{T}}}_{imp_\Sigma}$, conjunto soporte del género distinguido, la representación $\mathcal{R}^{rep,\underline{\mathtt{T}},can}_{imp_\Sigma}$ que vamos a definir se basa en almacenar los objetos funcionales procedentes de las implementaciones del TAD de partida, algunos de los cuales provienen de los programas que implementan a las operaciones de Σ y el resto de los que implementan a los invariantes e igualdades de las representaciones sobre las que se han definido las implementaciones. Para ello definimos una estructura con la idea de almacenar en sus campos todos los objetos funcionales que provienen de una implementación de $\mathcal{T}$ de la categoría $Imp^{\underline{\mathtt{T}}}_{Dec}(\mathcal{T})$. Así, definimos el siguiente tipo Common Lisp

```
(defstruct REP-IMP-g imp- rep-invar_g1 ... rep-invar_gn rep-igual_g1 ...
                      rep-igual_gn (:include IMP-g))
```

La opción `:include` añade a la estructura que se está definiendo todos los campos de la estructura a la que se refiere (para más detalles, ver [116], página 477).

Consideramos también el siguiente predicado Common Lisp

```
(defun DREP-can-p (x)
     (and (typep x 'REP-IMP-g)
          (functionp (REP-IMP-g-imp-σ1 x))
                     ...
          (functionp (REP-IMP-g-imp-σm x))
          (functionp (REP-IMP-g-rep-invar_g1 x))
                     ...
          (functionp (REP-IMP-g-rep-invar_gn x))
          (functionp (REP-IMP-g-rep-igual_g1 x))
                     ...
          (functionp (REP-IMP-g-rep-igual_gn x))))
```

predicado que decide si un objeto Common Lisp es de tipo `REP-IMP-g` y todos sus campos son objetos funcionales.

Apoyándonos en las dos definiciones anteriores para determinar el dominio $\mathcal{D}_{imp_\Sigma}^{rep,\underline{\mathtt{T}},can}$, lo definimos como sigue: $\mathcal{D}_{imp_\Sigma}^{rep,\underline{\mathtt{T}},can} =$ `(satisfies DREP-can-p)`.

Como datos representativos dentro del dominio anterior, es decir, como elementos del soporte de la representación, debemos considerar aquellos que provienen de implementaciones de la categoría $Imp_{Dec}^{\underline{\mathtt{T}}}(\mathcal{T})$, es decir, implementaciones con invariantes e igualdades de representación calculables y explícitos y programas totales sobre el producto de los soportes de las representaciones. Por tanto, si $\mathtt{x} \in \mathcal{D}_{imp_\Sigma}^{rep,\underline{\mathtt{T}},can}$, $\mathtt{x}$ está en el soporte $\mathcal{S}_{imp_\Sigma}^{rep,\underline{\mathtt{T}},can}$ si existe $\mathcal{J} = (\underline{\mathcal{R}}^{\mathcal{J}}, (\mathtt{p}_\delta^{\mathcal{J}})_{\delta\in\Delta_\Sigma}) \in Obj(Imp_{Dec}^{\underline{\mathtt{T}}}(\mathcal{T}))$ verificando que los códigos de los programas de $\mathcal{J}$ coinciden (por la igualdad `eq`, única razonable entre objetos funcionales) con los almacenados en los campos de $\mathtt{x}$. Formalmente, debe verificarse:

- `(eq` $\mathtt{c}_{invar_g}^{\mathcal{J}}$ `(REP-IMP-g-rep-invar`$_g$ `x))`, para cada $g \in G$, siendo $\mathtt{c}_{invar_g}^{\mathcal{J}}$ el objeto funcional del programa $\mathtt{p}_{invar_g}^{\mathcal{J}} = (\mathtt{c}_{invar_g}^{\mathcal{J}},\ \mathtt{T}_g,\ \mathtt{T}_g,\ \mathcal{U})$;
- `(eq` $\mathtt{c}_{igual_g}^{\mathcal{J}}$ `(REP-IMP-g-rep-igual`$_g$ `x))`, también para cada $g \in G$, y donde $\mathtt{c}_{igual_g}^{\mathcal{J}}$ es el código del programa $\mathtt{p}_{igual_g}^{\mathcal{J}} = (\mathtt{c}_{igual_g}^{\mathcal{J}},\ \mathcal{S}_g^{\mathcal{J}} \times \mathcal{S}_g^{\mathcal{J}},\ \mathtt{T}_g,\ \mathtt{T}_g,\ \mathcal{U})$, siendo $\mathcal{S}_g^{\mathcal{J}}$ el soporte de la representación $\mathcal{R}_g^{\mathcal{J}}$;
- `(eq` $\mathtt{c}_\sigma^{\mathcal{J}}$ `(REP-IMP-g-imp-`σ `x))`, para cada $(\sigma\colon g_1 \dots g_k \to g) \in \Omega$, siendo $\mathtt{p}_\sigma^{\mathcal{J}} = (\mathtt{c}_\sigma^{\mathcal{J}}, \mathcal{S}_{g_1}^{\mathcal{J}} \times \cdots \times \mathcal{S}_{g_k}^{\mathcal{J}},\ \mathtt{T}_{g_1}, \dots, \mathtt{T}_{g_k},\ \mathtt{T}_g)$ el programa que implementa al operador σ en la implementación $\mathcal{J}$.

Por definición de la categoría $Imp_{Dec}^{\underline{\mathtt{T}}}(\mathcal{T})$, para cada $g \in G$, la función definida por el programa $\mathtt{p}_{invar_g}^{\mathcal{J}}$, coincide con el invariante de la representación $\mathcal{R}_g^{\mathcal{J}}$, es decir, es la función característica del soporte $\mathcal{S}_g^{\mathcal{J}}$ en $\mathtt{T}_g$ y, por el mismo motivo, la función definida por el programa $\mathtt{p}_{igual_g}^{\mathcal{J}}$ coincide con la igualdad de la representación $\mathcal{R}_g^{\mathcal{J}}$, $=_g^{\mathcal{J}}$. Además, para cada $\sigma \in \Omega$, la función definida por el programa $\mathtt{p}_\sigma^{\mathcal{J}}$ es total sobre el producto de los soportes de las representaciones correspondientes.

La función de abstracción $\alpha_{imp_\Sigma}^{rep,\underline{\mathtt{T}},can}\colon \mathcal{S}_{imp_\Sigma}^{rep,\underline{\mathtt{T}},can} \to \mathcal{M}_{imp_\Sigma}^{rep,\underline{\mathtt{T}}}$ lleva cada registro, formado por objetos funcionales, a la tupla de funciones definidas por esos objetos funcionales sobre el producto de los soportes de las representaciones. Así, si $\mathtt{x} \in \mathcal{S}_{imp_\Sigma}^{rep,\underline{\mathtt{T}},can}$, la función

de abstracción se define como $\alpha^{rep,\underline{\mathtt{T}},can}_{imp_\Sigma}(\mathtt{x}) := (f^{\mathtt{x}}_\delta)_{\delta\in\Delta_\Sigma}$, donde las funciones de la tupla anterior se definen como sigue:

- para cada $g \in G$, la función $f^{\mathtt{x}}_{invar_g} : \mathtt{T}_g \to \mathcal{U}$ es total y definida, para cada $\mathtt{d} \in \mathtt{T}_g$, por $f^{\mathtt{x}}_{invar_g}(\mathtt{d}) :=$ `(funcall (REP-IMP-g-rep-invar`$_g$ `x) d)`. Dado $\mathtt{x} \in \mathcal{S}^{rep,\underline{\mathtt{T}},can}_{imp_\Sigma}$, por la propia definición de este soporte, se tiene que `(REP-IMP-g-rep-invar`$_g$ `x)` es un objeto funcional tal que su evaluación sobre objetos de $\mathtt{T}_g$ siempre termina. Por tanto, la función $f^{\mathtt{x}}_{invar_g}$ está bien definida. El objeto funcional `(REP-IMP-g-rep-invar`$_g$ `x)` define un subconjunto $S^{\mathtt{x}}_g$ de $\mathtt{T}_g$ que coincide con el soporte de la representación $\mathcal{R}^{\mathcal{J}}_g$

$$S^{\mathtt{x}}_g = \{\mathtt{d} \in \mathtt{T}_g \mid \texttt{(not (eq (funcall (REP-IMP-g-rep-invar}_g\ \texttt{x) d) nil))}\}$$

- para cada $g \in G$, la función $f^{\mathtt{x}}_{igual_g} : \mathtt{T}_g \times \mathtt{T}_g \to \mathcal{U}$ tiene por dominio $S^{\mathtt{x}}_g \times S^{\mathtt{x}}_g$ y se define como sigue: para cada $\mathtt{d1}, \mathtt{d2} \in S^{\mathtt{x}}_g$, $f^{\mathtt{x}}_{igual_g}(\mathtt{d1}, \mathtt{d2}) :=$ `(funcall (REP-IMP-g-rep-igual`$_g$ `x) d1 d2)`;
- para cada $(\sigma\colon \omega \to v) \in \Omega$ con $\omega = g_1 \dots g_k$, la función $f^{\mathtt{x}}_\sigma\colon \mathtt{T}_\omega \to \mathtt{T}_v$ tiene por dominio $S^{\mathtt{x}}_\omega$, siendo $S^{\mathtt{x}}_\omega = S^{\mathtt{x}}_{g_1} \times \cdots \times S^{\mathtt{x}}_{g_k}$, y viene definida por $f^{\mathtt{x}}_\sigma(\mathtt{d1}, \dots, \mathtt{dk}) :=$ `(funcall (REP-IMP-g-imp-`σ `x) d1`$\dots$`dk )`, para todo $(\mathtt{d1}, \dots, \mathtt{dk}) \in S^{\mathtt{x}}_\omega$.

La función de abstracción está bien definida ya que si $\mathtt{x} \in \mathcal{S}^{rep,\underline{\mathtt{T}},can}_{imp_\Sigma}$, entonces su imagen por la función de abstracción, $(f^{\mathtt{x}}_\delta)_{\delta\in\Delta_\Sigma}$, está en $\mathcal{M}^{rep,\underline{\mathtt{T}}}_{imp_\Sigma}$, puesto que la implementación $\mathcal{J} = (\underline{\mathcal{R}}^{\mathcal{J}}, (\mathtt{p}^{\mathcal{J}}_\delta)_{\delta\in\Delta_\Sigma}) \in Obj(Imp^{\underline{\mathtt{T}}}_{Dec}(\mathcal{T}))$ que hace que $\mathtt{x}$ esté en el soporte, lo garantiza.

Para terminar la definición de la representación $\mathcal{R}^{rep,\underline{\mathtt{T}},can}_{imp_\Sigma}$, queda por definir la igualdad de la representación. Tomamos como igualdad de representación $=_{\mathcal{R}^{rep,\underline{\mathtt{T}},can}_{imp_\Sigma}}$, la de la abstracción.

- Para concluir la definición de la implementación $\mathcal{I}^{rep,\underline{\mathtt{T}},can}$ construimos los programas que implementan a las operaciones de $\mathcal{M}^{rep,\underline{\mathtt{T}}}$:

  - para cada género $g \in G$, un programa que implementa a la operación $rep\_invar_{\mathcal{M}^{rep,\underline{\mathtt{T}}}_g} : \mathcal{M}^{rep,\underline{\mathtt{T}}}_{imp_\Sigma} \times \mathtt{T}_g \to \mathcal{U}$ es el programa $\mathtt{p}^{rep,\underline{\mathtt{T}},can}_{rep\_invar_g} = (\mathtt{c}^{\mathtt{p}^{rep,\underline{\mathtt{T}},can}_{rep\_invar_g}}, \mathcal{S}^{rep,\underline{\mathtt{T}},can}_{imp_\Sigma} \times \mathtt{T}_g,\ \mathcal{D}^{rep,\underline{\mathtt{T}},can}_{imp_\Sigma},\ \mathtt{T}_g,\ \mathcal{U})$, siendo el código del programa el siguiente objeto funcional

    ```
    c^{p^{rep,T,can}_{rep_invar_g}} ≡ #'(lambda (x d)
                                          (funcall (REP-IMP-g-rep-invar_g x) d))
    ```

    Cada uno de estos programas determina una familia de subconjuntos de $\mathtt{T}_g$, uno por cada $\mathtt{x}$ del soporte $\mathcal{S}^{rep,\underline{\mathtt{T}},can}_{imp_\Sigma}$, subconjuntos utilizados para definir el soporte de los programas que implementan al resto de operaciones. Así, para cada $g \in G$ y cada $\mathtt{x} \in \mathcal{S}^{rep,\underline{\mathtt{T}},can}_{imp_\Sigma}$, consideramos el conjunto $S^{\mathtt{x}}_g$, ya definido anteriormente del siguiente modo

$$S^{\mathtt{x}}_g = \{\mathtt{d} \in \mathtt{T}_g \mid \texttt{(not (eq (funcall (REP-IMP-g-rep-invar}_g\ \texttt{x) d) nil))}\}$$

· para cada género $g \in G$, la operación $rep\_igual_{\mathcal{M}_g^{rep,\underline{\mathtt{T}}}}\colon \mathcal{M}_{imp_\Sigma}^{rep,\underline{\mathtt{T}}} \times \mathtt{T}_g \times \mathtt{T}_g \to \mathcal{U}$ es implementada por el programa $\mathtt{p}_{rep\_igual_g}^{rep,\underline{\mathtt{T}},can}$ con soporte

$$\mathtt{S}^{\mathtt{p}_{rep\_igual_g}^{rep,\underline{\mathtt{T}},can}} = \left\{(\mathtt{x},\mathtt{d1},\mathtt{d2}) \in \mathcal{S}_{imp_\Sigma}^{rep,\underline{\mathtt{T}},can} \times \mathtt{T}_g \times \mathtt{T}_g \;\mid\; \mathtt{d1},\mathtt{d2} \in S_g^{\mathtt{x}}\right\}$$

y siendo su código el siguiente objeto funcional

$$\mathtt{c}^{\mathtt{p}_{rep\_igual_g}^{rep,\underline{\mathtt{T}},can}} \equiv \mathtt{\#'(lambda\ (x\ d1\ d2)}\ (\mathtt{funcall\ (REP\text{-}IMP\text{-}g\text{-}rep\text{-}igual}_g\ \mathtt{x)\ d1\ d2))}$$

· para cada $\sigma\colon g_1 \dots g_k \to g$ operación de Ω, la función $imp\_\sigma_{\mathcal{M}^{rep,\underline{\mathtt{T}}}}\colon \mathcal{M}_{imp_\Sigma}^{rep,\underline{\mathtt{T}}} \times \mathtt{T}_{g_1} \times \cdots \times \mathtt{T}_{g_k} \to \mathtt{T}_g$, se implementa por medio del programa $\mathtt{p}_{imp\_\sigma}^{rep,\underline{\mathtt{T}},can} = (\mathtt{c}^{\mathtt{p}_{imp\_\sigma}^{rep,\underline{\mathtt{T}},can}}, \mathtt{S}^{\mathtt{p}_{imp\_\sigma}^{rep,\underline{\mathtt{T}},can}}, \mathcal{D}_{imp_\Sigma}^{rep,\underline{\mathtt{T}},can}, \mathtt{T}_{g_1}, \dots, \mathtt{T}_{g_k}, \mathtt{T}_g)$ cuyo soporte viene definido por

$$\mathtt{S}^{\mathtt{p}_{imp\_\sigma}^{rep,\underline{\mathtt{T}},can}} = \left\{(\mathtt{x},\mathtt{d1},\dots,\mathtt{dk}) \in \mathcal{S}_{imp_\Sigma}^{rep,\underline{\mathtt{T}},can} \times \mathtt{T}_{g_1} \times \cdots \times \mathtt{T}_{g_k} \mid (\mathtt{d1},\dots,\mathtt{dk}) \in S_{g_1}^{\mathtt{x}} \times \cdots \times S_{g_k}^{\mathtt{x}}\right\}$$

y siendo el código del programa el siguiente objeto funcional

$$\mathtt{c}^{\mathtt{p}_{imp\_\sigma}^{rep,\underline{\mathtt{T}},can}} \equiv \mathtt{\#'(lambda\ (x\ d1\ \dots\ dk)}\ (\mathtt{funcall\ (REP\text{-}IMP\text{-}g\text{-}imp\text{-}}\sigma\ \mathtt{x)\ d1\ \dots\ dk))}$$

En la definición de cada implementación $\mathcal{I}^{rep,\underline{\mathtt{T}},can}$ la parte más delicada es la definición de la función de abstracción, ya que supone obtener a partir de una tupla formada por objetos funcionales una tupla de conjuntos, siendo cada uno de ellos el dominio de definición asociado a uno de esos objetos funcionales. Esto ha sido posible porque como los tipos sobre los que se aplican los objetos funcionales están fijados, asociando a cada objeto funcional un dominio tenemos una tupla de programas que, por las condiciones impuestas en la definición del soporte $\mathcal{S}_{imp_\Sigma}^{rep,\underline{\mathtt{T}},can}$ coinciden con los programas de alguna implementación del TAD de partida. Vamos a precisar un poco más este aspecto, ya que puede resultar confuso.

Cada tupla $\mathtt{x}$ de objetos funcionales perteneciente al soporte $\mathcal{S}_{imp_\Sigma}^{rep,\underline{\mathtt{T}},can}$ contiene, para cada género g de Σ, un objeto funcional que permite determinar un subconjunto del tipo $\mathtt{T}_g$, utilizado para asociar los dominios de definición al resto de programas. Por definición del soporte $\mathcal{S}_{imp_\Sigma}^{rep,\underline{\mathtt{T}},can}$, el objeto funcional almacenado en la componente $\mathtt{(REP\text{-}IMP\text{-}g\text{-}rep\text{-}invar}_g\ \mathtt{x)}$ es un objeto cuyas llamadas terminan para cualquier dato del tipo $\mathtt{T}_g$, es decir, es total sobre $\mathtt{T}_g$ ya que es una implementación del invariante de una representación con dominio $\mathtt{T}_g$. Por ello, al objeto funcional almacenado en el campo $\mathtt{(REP\text{-}IMP\text{-}g\text{-}rep\text{-}invar}_g\ \mathtt{x)}$ se le puede asociar como soporte el tipo $\mathtt{T}_g$. Además, cada objeto funcional $\mathtt{(REP\text{-}IMP\text{-}g\text{-}rep\text{-}invar}_g\ \mathtt{x)}$ permite definir un subconjunto de $\mathtt{T}_g$, subconjunto formado por aquellos objetos sobre los que la evaluación del objeto funcional no devuelve `nil`, y que, por definición del soporte $\mathcal{S}_{imp_\Sigma}^{rep,\underline{\mathtt{T}},can}$, debe coincidir con el soporte de la representación sobre la que se define una implementación que hace que $\mathtt{x}$ esté en el soporte. Hemos denotado por $S_g^{\mathtt{x}}$ a este subconjunto de $\mathtt{T}_g$. Así, para cada tupla $\mathtt{x}$ de objetos funcionales del soporte $\mathcal{S}_{imp_\Sigma}^{rep,\underline{\mathtt{T}},can}$, utilizando estos subconjuntos $\{S_g^{\mathtt{x}}\}_{g\in G}$ podemos asociar dominios al resto de los objetos funcionales almacenados en $\mathtt{x}$. Para cada $g \in G$, el objeto funcional almacenado en el campo $\mathtt{(REP\text{-}IMP\text{-}g\text{-}rep\text{-}igual}_g\ \mathtt{x)}$ debe tener a $S_g^{\mathtt{x}} \times S_g^{\mathtt{x}}$ como dominio puesto que debe implementar a la igualdad de representación que proviene de la misma representación que el invariante. Para cada operación $\sigma\colon g_1 \dots g_k \to g$ de Ω, el dominio asociado al campo $\mathtt{(REP\text{-}IMP\text{-}g\text{-}imp\text{-}}\sigma\ \mathtt{x)}$ de $\mathtt{x}$, por definición del soporte $\mathcal{S}_{imp_\Sigma}^{rep,\underline{\mathtt{T}},can}$, debe coincidir con el producto $S_{g_1}^{\mathtt{x}} \times \cdots \times S_{g_k}^{\mathtt{x}}$. Esta condición impuesta desde el principio es natural si

pensamos que al estar almacenando explícitamente los soportes de las representaciones sobre las que reposa cada implementación, es natural que todos los elementos de los soportes de las representaciones sean significativos para los programas. Así, en el marco del capítulo anterior, a partir de una tupla de objetos funcionales, no disponemos de ninguna información sobre los dominios de definición de estos objetos funcionales. Por ello, la única forma de asociar a cada objeto funcional un dominio es fijándolo de antemano, lo que nos ha obligado a fijar una tupla de conjuntos, uno por cada operación de la signatura de partida, conjuntos que juegan el papel de dominios de definición de los programas. En este caso, a diferencia del anterior, no es necesario fijar explícitamente los dominios de definición de las operaciones, sino que estos dominios son definidos a partir de los programas que implementan a los invariantes de las representaciones. Así, el papel jugado en el capítulo anterior por el Ω-conjunto $\underline{\mathtt{S}}$ formado por los dominios de definición de las operaciones, lo juegan en este caso los conjuntos $S_g^{\mathtt{x}}$, para cada $g \in G$ y, cada tupla $\mathtt{x}$ de objetos funcionales que proviene de una implementación de la categoría $Imp_{Dec}^{\underline{\mathtt{T}}}(\mathcal{T})$.

Comparando las situaciones de ambos capítulos, podemos ver que el problema radica en asociar dominios de definición a los objetos funcionales que representan implementaciones. Ahora tenemos dos formas de hacerlo. Esta segunda forma, permite incluir en la misma familia implementaciones cuyos programas posean distintos dominios de definición, ya que en este caso cada tupla contiene implícitamente subconjuntos que pueden ser diferentes de unas tuplas a otras y, este caso quedaba excluido en el capítulo anterior. A cambio, estamos obligados a exigir totalidad sobre determinados conjuntos definidos a partir de los anteriores (los conjuntos $S_g^{\mathtt{x}}$).

Por otro lado, al recoger en la representación en la máquina de una implementación las igualdades de las representaciones, estamos incluyendo un paso de abstracción, paso que en el capítulo anterior quedaba totalmente fuera de la máquina. De esta forma, a partir del mismo conjunto de objetos Lisp, es decir, fijado un tipo, considerando distintas relaciones de igualdad, podemos codificar distintos conjuntos.

La siguiente proposición garantiza que tal y como la hemos definido, $\mathcal{I}^{rep,\underline{\mathtt{T}},can}$ es realmente una implementación de $\mathcal{M}^{rep,\underline{\mathtt{T}}}$.

Proposición 4.2.6 *Cada objeto $\mathcal{I}^{rep,\underline{\mathtt{T}},can}$ es una implementación de la $\Sigma_{imp_{rep}}$-álgebra $\mathcal{M}^{rep,\underline{\mathtt{T}}}$.*

La demostración del resultado recogido en la proposición anterior sigue el mismo esquema que la de la proposición 3.7.7 del capítulo anterior. En ella se utilizan de forma esencial: el acceso a un campo de un `defstruct`; la elección de las representaciones literales para los géneros de Σ y la representación de $\mathcal{U}$ que interpreta cualquier objeto Lisp distinto de `nil` como el valor booleano *true*; la condición sobre la existencia de una implementación de la categoría $Imp_{Dec}^{\underline{\mathtt{T}}}(\mathcal{T})$ para que una tupla de objetos funcionales (formada por los códigos que provienen de los programas de esta implementación) pertenezca al soporte de la representación del género imp_Σ (sobre esto último, destacar especialmente la posibilidad de determinar los soportes de las representaciones de la implementación de $Imp_{Dec}^{\underline{\mathtt{T}}}(\mathcal{T})$, así, como la condición de totalidad sobre los soportes de las representaciones exigida a los programas que provienen de esa implementación); la elección de la igualdad de la abstracción como igualdad de la representación que permite demostrar compatibilidad y coherencia con las representaciones de los programas de $\mathcal{I}^{rep,\underline{\mathtt{T}},can}$; y, la descomposición del dominio de cada programa de $\mathcal{I}^{rep,\underline{\mathtt{T}},can}$ como

$$\bigcup_{\mathtt{x} \in \mathcal{S}_{imp_\Sigma}^{rep,\underline{\mathtt{T}},can}} \left(\{\mathtt{x}\} \times S_\omega^{\mathtt{x}}\right)$$

siendo $S_\omega^{\mathtt{x}}$ el producto de los soportes de las representaciones correspondientes a los géneros que

aparecen en la aridad de la operación que se está implementando.

Las implementaciones de la familia $\{\mathcal{I}^{rep,\underline{\mathtt{T}},can}\}_{\underline{\mathtt{T}} \text{ es } G-\text{tipo}}$ verifican propiedades universales en el sentido de la Teoría de Categorías, del mismo tipo que las esgrimidas en el capítulo anterior, motivo por el que también las denominamos *implementaciones canónicas*.

4.2.3 Categorías de implementaciones

Para obtener las propiedades que caracterizan a las implementaciones del tipo $\mathcal{I}^{rep,\underline{\mathtt{T}},can}$, vamos a considerar una adecuada subcategoría de la categoría de implementaciones de $\mathcal{M}^{rep,\underline{\mathtt{T}}}$. Del mismo modo que hemos hecho en el capítulo anterior, para definir esta subcategoría vamos a apoyarnos en algunas propiedades que verifica la implementación $\mathcal{I}^{rep,\underline{\mathtt{T}},can}$ y que destacamos a continuación:

- Para cada $g \in G$ se ha tomado la representación literal sobre el tipo $\mathtt{T}_g$; para el género *bool* se ha tomado la que interpreta cualquier objeto Lisp distinto de `nil` con el valor booleano *true*.
- Cada dato $\mathtt{x}$ del soporte de la representación del género imp_Σ, determina una familia de conjuntos $\{S_g^{\mathtt{x}}\}_{g\in G}$, con $S_g^{\mathtt{x}} \subseteq \mathtt{T}_g$ para cada $g \in G$. Estos conjuntos se utilizan para definir los dominios de las funciones que están en la imagen de $\mathtt{x}$ por la función de abstracción, del siguiente modo: si $\mathtt{x} \in \mathcal{S}_{imp_\Sigma}^{rep,\underline{\mathtt{T}},can}$ y $\alpha_{imp_\Sigma}^{rep,\underline{\mathtt{T}},can}(\mathtt{x}) := (f_\delta)_{\delta\in\Delta_\Sigma}$ se verifican
  - f_{invar_g} es total para todo $g \in G$;
  - $Def(f_{igual_g}) = S_g^{\mathtt{x}} \times S_g^{\mathtt{x}}$, para todo $g \in G$;
  - $Def(f_\sigma) = S_\omega^{\mathtt{x}}$, para todo $(\sigma\colon \omega \to v) \in \Omega$.
- Además, los dominios de los programas que implementan a las operaciones de $\mathcal{M}^{rep,\underline{\mathtt{T}}}$ están determinados por los conjuntos anteriores verificándose las siguientes propiedades:
  - para cada $g \in G$, el programa que implementa al operador $rep\_invar_{\mathcal{M}_g^{rep,\underline{\mathtt{T}}}}$ es total;
  - para cada $g \in G$, el dominio del programa que implementa al operador $rep\_igual_{\mathcal{M}_g^{rep,\underline{\mathtt{T}}}}$ puede expresarse del siguiente modo
$$\bigcup_{\mathtt{x}\in\mathcal{S}_{imp_\Sigma}^{rep,\underline{\mathtt{T}},can}} \left(\{\mathtt{x}\} \times S_g^{\mathtt{x}} \times S_g^{\mathtt{x}}\right)$$
  - para cada $(\sigma\colon \omega \to v) \in \Omega$, el dominio del programa que implementa al operador $imp\_\sigma_{\mathcal{M}^{rep,\underline{\mathtt{T}}}}$ puede describirse como sigue
$$\bigcup_{\mathtt{x}\in\mathcal{S}_{imp_\Sigma}^{rep,\underline{\mathtt{T}},can}} \left(\{\mathtt{x}\} \times S_\omega^{\mathtt{x}}\right)$$

Apoyándonos en las propiedades anteriores que verifica cada implementación $\mathcal{I}^{rep,\underline{\mathtt{T}},can}$ de $\mathcal{M}^{rep,\underline{\mathtt{T}}}$, denotamos por $Imp_{rep}^{\underline{\mathtt{T}},\{\}}(\mathcal{M}^{rep,\underline{\mathtt{T}}})$ a la subcategoría de la categoría de implementaciones de $\mathcal{M}^{rep,\underline{\mathtt{T}}}$ con objetos aquellas implementaciones de $\mathcal{M}^{rep,\underline{\mathtt{T}}}$, $\mathcal{I} = \left(\underline{\mathcal{R}^{\mathcal{I}}}, (\mathtt{p}^{\mathcal{I}}_{rep\_invar_g})_{g\in G}, (\mathtt{p}^{\mathcal{I}}_{rep\_igual_g})_{g\in G}, (\mathtt{p}^{\mathcal{I}}_{imp\_\sigma})_{\sigma\in\Omega}\right)$, que verifican las siguientes condiciones:

i) Para cada $g \in G$, la representación para el tipo $\mathtt{T}_g$ es la literal; también para el tipo $\mathcal{U}$.

ii) Para cada $g \in G$, el programa $\mathtt{p}^{\mathcal{I}}_{rep\_invar_g}$ es total sobre el producto de los soportes de las representaciones, es decir, tiene como dominio $\mathcal{S}^{\mathcal{I}}_{imp_\Sigma} \times \mathtt{T}_g$. Esta condición hace posible la definición, para cada $\mathtt{x} \in \mathcal{S}^{\mathcal{I}}_{imp_\Sigma}$, del siguiente conjunto

$$S_g^{\mathtt{x},\mathcal{I}} = \left\{ \mathtt{d} \in \mathtt{T}_g \ \mid \ \texttt{(not (eq (funcall c}^{\mathcal{I}}_{rep\_invar_g}\texttt{ x d) nil))} \right\}$$

conjunto que vamos a exigir determine los dominios del resto de programas tal y como se recoge en el siguiente punto.

iii) Los dominios de definición del resto de programas son los siguientes:

· para cada $g \in G$, el dominio del programa $\mathtt{p}^{\mathcal{I}}_{rep\_igual_g}$ es el siguiente conjunto

$$Def(\mathtt{p}^{\mathcal{I}}_{rep\_igual_g}) = \left\{ (\mathtt{x}, \mathtt{d1}, \mathtt{d2}) \in \mathcal{S}^{\mathcal{I}}_{imp_\Sigma} \times \mathtt{T}_g \times \mathtt{T}_g \ \mid \ \mathtt{d1}, \mathtt{d2} \in S_g^{\mathtt{x},\mathcal{I}} \right\}$$

· para cada $(\sigma\colon \omega \to v) \in \Omega$, el dominio del programa $\mathtt{p}^{\mathcal{I}}_{imp\_\sigma}$ viene dado por

$$Def(\mathtt{p}^{\mathcal{I}}_{imp\_\sigma}) = \left\{ (\mathtt{x}, \mathtt{d}_\omega) \in \mathcal{S}^{\mathcal{I}}_{imp_\Sigma} \times \mathtt{T}_\omega \ \mid \ \mathtt{d}_\omega \in S_\omega^{\mathtt{x},\mathcal{I}} \right\}$$

Como morfismos de la categoría $Imp^{\underline{\mathtt{T}},\{\}}_{rep}(\mathcal{M}^{rep,\underline{\mathtt{T}}})$ se toman aquellas i–transformaciones entre implementaciones que son la identidad sobre la parte de los modelos y sobre los soportes de todas las representaciones excepto sobre la del género imp_Σ.

Cada implementación canónica $\mathcal{I}^{rep,\underline{\mathtt{T}},can}$ pertenece a la categoría $Imp^{\underline{\mathtt{T}},\{\}}_{rep}(\mathcal{M}^{rep,\underline{\mathtt{T}}})$.

A partir de cada implementación $\mathcal{I}$ de la categoría $Imp^{\underline{\mathtt{T}}}_{Dec}(\mathcal{T})$ podemos definir de manera natural una implementación $\mathcal{I}^*$ de la $\Sigma_{imp_{rep}}$-álgebra $\mathcal{M}^{rep,\underline{\mathtt{T}}}$, lo que permite ver cada implementación de $Imp^{\underline{\mathtt{T}}}_{Dec}(\mathcal{T})$ como dato de $\mathcal{I}^{rep,\underline{\mathtt{T}},can}$: $\mathcal{I}^*$ se define del mismo modo que la implementación canónica $\mathcal{I}^{rep,\underline{\mathtt{T}},can}$ diferenciándose de ésta en el soporte de la representación correspondiente al género distinguido $\mathcal{S}^{\mathcal{I}^*,rep,\underline{\mathtt{T}}}_{imp_\Sigma}$, que se define, en el caso de $\mathcal{I}^*$, como el conjunto formado por un único objeto de tipo `REP-IMP-g` que es el que almacena en sus campos los códigos de los programas de la implementación $\mathcal{I}$. Definida de este modo, $\mathcal{I}^*$ es una implementación de $\mathcal{M}^{rep,\underline{\mathtt{T}}}$. La definición de una implementación $\mathcal{I}^*$ de $\mathcal{M}^{rep,\underline{\mathtt{T}}}$ a partir de la implementación $\mathcal{I}$ de $\mathcal{T}$ se extiende a un funtor inclusión entre las categorías $Imp^{\underline{\mathtt{T}}}_{Dec}(\mathcal{T})$ e $Imp^{\underline{\mathtt{T}},\{\}}_{rep}(\mathcal{M}^{rep,\underline{\mathtt{T}}})$. Esta propiedad unida al hecho de que cada implementación canónica $\mathcal{I}^{rep,\underline{\mathtt{T}},can}$ recoge los códigos de los programas de *todas* las implementaciones de $Imp^{\underline{\mathtt{T}}}_{Dec}(\mathcal{T})$, y nada más, permite enunciar el resultado recogido en el siguiente teorema.

Teorema 4.2.7 *La implementación $\mathcal{I}^{rep,\underline{\mathtt{T}},can}$ es el coproducto en $Imp^{\underline{\mathtt{T}},\{\}}_{rep}(\mathcal{M}^{rep,\underline{\mathtt{T}}})$ de las implementaciones de la imagen del funtor inclusión de la categoría de implementaciones de $\mathcal{T}$ $Imp^{\underline{\mathtt{T}},\{\}}_{Dec}(\mathcal{T})$, en $Imp^{\underline{\mathtt{T}},\{\}}_{rep}(\mathcal{M}^{rep,\underline{\mathtt{T}}})$. Es decir, $\mathcal{I}^{rep,\underline{\mathtt{T}},can}$ es el coproducto en $Imp^{\underline{\mathtt{T}},\{\}}_{rep}(\mathcal{M}^{rep,\underline{\mathtt{T}}})$ de la familia $\{\mathcal{I}^*\}_{\mathcal{I} \in Obj(Imp^{\underline{\mathtt{T}},\{\}}_{Dec}(\mathcal{T}))}$.*

De esta forma cada implementación canónica es el mínimo objeto que recoge a todas las implementaciones de la categoría $Imp^{\underline{\mathtt{T}},\{\}}_{Dec}(\mathcal{T})$. Además, cada implementación canónica posee otra propiedad universal que recogemos a continuación.

Teorema 4.2.8 *La implementación canónica $\mathcal{I}^{rep,\underline{\mathtt{T}},can}$ es objeto final en la categoría $Imp_{rep}^{\underline{\mathtt{T}},\{\}}(\mathcal{M}^{rep,\underline{\mathtt{T}}})$.*

La demostración de este resultado es totalmente análoga a la dada para el resultado correspondiente en el capítulo anterior. En ella se construye directamente la i–transformación y la parte dura consiste en probar que la imagen de un dato del soporte de la representación de partida, por la i–transformación, cae en el soporte de la representación canónica.

Si denotamos por $\mathcal{T}_{imp_{rep}}$ al TAD formado por todos los modelos $\mathcal{M}^{rep,\underline{\mathtt{T}}}$, para cada G-tipo $\underline{\mathtt{T}}$, y denotamos por $Imp_{rep}^{\{\}}(\mathcal{T}_{imp_{rep}})$ a la categoría de implementaciones obtenida por agrupación de las categorías $Imp_{rep}^{\underline{\mathtt{T}},\{\}}(\mathcal{M}^{rep,\underline{\mathtt{T}}})$, el siguiente resultado extiende el anterior.

Corolario 4.2.9 *La familia de implementaciones canónicas $\{\mathcal{I}^{rep,\underline{\mathtt{T}},can}\}_{\underline{\mathtt{T}} \text{ es } G-\text{tipo}}$ es una familia final en la categoría $Imp_{rep}^{\{\}}(\mathcal{T}_{imp_{rep}})$.*

De modo análogo a lo visto en el capítulo anterior, trabajando únicamente con r–transformaciones calculables, los resultados anteriores se trasladan a ese marco.

De los resultados anteriores podemos deducir que la implementación canónica es la más general entre todas las implementaciones de $\mathcal{M}^{rep,\underline{\mathtt{T}}}$. En este sentido, desde cualquier otra implementación de $\mathcal{M}^{rep,\underline{\mathtt{T}}}$ existe un paso natural, dado por el morfismo final, a una implementación canónica.

A la vista de lo anterior, y ya en otra línea, se ve que el caso general tal y como se ha presentado es bastante restrictivo puesto que partimos de implementaciones del TAD de partida que verifican determinadas condiciones, concretamente implementaciones con G-tipo fijado, soportes e igualdades de representación calculables y explícitos y soportes de los programas totales sobre el producto correspondiente de los soportes de las representaciones. Esto hace que solo modelemos implementaciones muy concretas. Sin embargo, debemos ir un poco más allá y tener en cuenta que el trasfondo en las condiciones anteriores está en ser capaces de asociar dominios de definición a códigos para obtener programas que formen parte de las implementaciones que estamos modelando. En el capítulo anterior lo que hemos hecho ha sido fijarlos. En éste avanzamos un poco más definiéndolos a partir de los invariantes de las representaciones. No obstante, sigue resultando bastante rígido ya que exigimos a los programas totalidad sobre el producto de los soportes de las representaciones. Sin embargo, lo esperable es que ante estructuraas concretas se disponga de información que permita determinar los dominios de definición. Así, podremos encontrarnos casos particulares en los que, por ejemplo, a partir de los soportes de las representaciones podamos deducir los dominios de los programas sin exigir totalidad sobre ellos. Por ejemplo, si pensamos en el caso de los conjuntos simpliciales sin fijar el conjunto de símplices geométricos, es claro que la condición de parcialidad que debe verificar el operador cara en cuanto a los índices correspondientes a la dimensión y al operador cara aplicado (éste último debe ser menor o igual que la dimensión del símplice sobre el que se aplica) no es recogida por lo tratado hasta aquí, ya que los dominios no coinciden con los soportes de las representaciones. Sin embargo, también es claro que, a partir de los soportes de las representaciones y teniendo en cuenta la relación existente entre los dos índices, se puede determinar el dominio del programa. En el caso de los complejos de cadenas, la situación es parecida, aunque el origen es diferente. En este caso, también hay una condición de parcialidad que no proviene del modelo, como ocurre en el caso de los conjuntos simpliciales, sino que proviene de la graduación, de haber utilizado un único género para los elementos de todos los grupos que forman el complejo de cadenas. Sin fijar el conjunto de generadores, pero disponien-

do del invariante y de la igualdad de la representación asociada al género en el que incluimos los elementos de todos los grupos que forman el complejo de cadenas, los soportes de los programas que implementan a las operaciones *opp*, *zero*, *mult* y *dffr* coinciden con los productos de los soportes de las representaciones correspondientes. Sin embargo, para el operador *suma* no ocurre lo mismo ya que la graduación da lugar a una condición de parcialidad: solo pueden sumarse elementos que estén en el mismo grado. Con lo anterior, pese a no ser total sobre los soportes de las representaciones, está claro cual debe ser el dominio del operador *suma*.

Como conclusión, el punto clave está en ser capaces de asociar un dominio de definición a cada uno de los códigos recogidos en la implementación canónica. En el capítulo anterior, lo hicimos fijando esos dominios. En éste, definiéndolos a partir de los soportes de las representaciones, en concreto, tomando dominios que son los productos de los correspondientes soportes. Pero los comentarios anteriores nos llevan a que en algunos casos, es posible definirlos apoyándonos en los soportes de las representaciones y en otras condiciones, sin necesidad de que sean los soportes de las representaciones los que los determinen totalmente. Podemos ir un poco más allá y pensar en que el código correspondiente al invariante almacenado en una tupla del objeto final no corresponda con el invariante de la representación sino con un invariante de otra cosa, que sirva de apoyo para asociar dominios a los códigos. Un ejemplo de esto último será disponer, en el caso de los conjuntos simpliciales de un invariante para los símplices geométricos, mientras que la situación análoga en el caso de los complejos de cadenas es disponer del invariante de los generadores. En determinadas condiciones, en ambos casos, siguiendo un patrón o estrategia somos capaces de determinar los dominios de los programas.

Para aclarar todo lo anterior incluimos a continuación algunos ejemplos cercanos a EAT que lo muestran. En este punto hemos elegido trabajar únicamente con complejos de cadenas entendiendo que el tratamiento de los conjuntos simpliciales es totalmente análogo y no aporta nada nuevo en este asunto. Hasta ahora siempre hemos tratado ambos ejemplos porque intervenía en ellos el diferente origen de la parcialidad (el tipo de los conjuntos simpliciales posee una parcialidad heredada de la del modelo matemático, mientras que la parcialidad en el caso de los complejos de cadenas proviene de la graduación). Sin embargo, para lo que sigue, ambos ejemplos nos permiten la misma capacidad expresiva por lo que a partir de ahora nos ocupamos únicamente de los complejos de cadenas.

4.2.4 Aplicación al Cálculo Simbólico: Complejos de cadenas

En este caso pretendemos modelar implementaciones del TAD $\mathcal{T}_{\mathtt{CC}}$ de los complejos de cadenas (ver su definición en la página 116), recogiendo también parte de las representaciones subyacentes a las implementaciones.

A partir de una determinada categoría de implementaciones del TAD de partida, $\mathcal{T}_{\mathtt{CC}}$ en este caso, se define un objeto que recoge a todas esas implementaciones.

Pensando en la representación de cualquier elemento de cualquier $\mathbb{Z}$-módulo de un complejo de cadenas como una combinación, consideramos los tipos $\mathtt{T}_{cmbn}$ e `integer` para los géneros *cmbn* e *int*, respectivamente. Recordar que los datos de tipo $\mathtt{T}_{cmbn}$ son los del conjunto

$$\mathtt{T}_{cmbn} = \Big\{ \big\langle \mathtt{p}, [(\mathtt{t}_1, \mathtt{x}_1), \ldots, (\mathtt{t}_\mathtt{m}, \mathtt{x}_\mathtt{m})] \big\rangle \;\mid\; \mathtt{p} \in \mathtt{integer},\ \mathtt{m} \in \mathtt{integer},\ \mathtt{(>= m\ 0)},\ \mathtt{t_i} \in \mathtt{integer} \text{ con } \mathtt{(not\ (=\ t_i\ 0))} \text{ para todo } \mathtt{i} = 1, \ldots, \mathtt{m}, \text{ y } \mathtt{(not\ (eq\ \ x_i\ x_j))} \text{ para todo } \mathtt{i}, \mathtt{j} \in \{1, \ldots, \mathtt{m}\} \text{ con } \mathtt{(not\ (=\ i\ j))} \Big\}$$

donde $\langle \mathtt{p}, [(\mathtt{t_1}, \mathtt{x_1}), \ldots, (\mathtt{t_m}, \mathtt{x_m})] \rangle$ es la representación externa que utilizamos en este marco para denotar a un dato de tipo `cmb` (recordar la definición del tipo `cmb`: `(defstruct cmb dgr lst)`). Además la igualdad del tipo $\mathtt{T}_{cmbn}$ identifica aquellas combinaciones con los mismos monomios, independientemente del orden en el que éstos aparezcan en la lista de datos de tipo `mnm` del campo `lst` (recordar que el tipo `mnm` se ha definido como `(defstruct mnm cff gnr)`).

Como ya hemos visto en el capítulo anterior, un buen punto de partida es considerar las implementaciones sobre representaciones naturales ya que éstas van a permitir trasladar de forma natural a las implementaciones algunas propiedades matemáticas de las álgebras a las que representan, y esa es la primera razón por la que hemos fijado los tipos anteriores. Por ejemplo, en el caso concreto de los complejos de cadenas con tipos los fijados, las representaciones naturales llevan a que existe un modo universal de definir el comportamiento de las operaciones de los grupos del complejo de cadenas, lo que permite no incluir en el objeto final sus códigos correspondientes puesto que no son necesarios para recuperar una implementación.

Con el objetivo de recoger parte de la representación subyacente a una implementación en lo que es la representación en la máquina de ésta, consideramos una subcategoría de la categoría de implementaciones del TAD $\mathcal{T}_{\mathtt{CC}}$. Según lo visto en la parte teórica de este capítulo, se trata de considerar implementaciones que permitan determinar, a partir de los invariantes, los dominios de definición de los programas. En el capítulo anterior, en el que fijábamos de antemano los dominios de los programas, hemos visto que en el caso de los complejos de cadenas el dominio de cualquier programa que implemente al operador *suma* está condicionado por el *grado* del grupo al que los elementos pertenecen. Consideramos la categoría que denotamos por $Imp^{\mathtt{T}^{nat}}_{Dec}(\mathcal{T}_{\mathtt{CC}})$, definida como la subcategoría plena de la categoría de implementaciones de $\mathcal{T}_{\mathtt{CC}}$ con objetos aquellas implementaciones con G-tipo $\underline{\mathtt{T}}$ sobre representaciones naturales de los tipos `integer` y $\mathtt{T}_{cmbn}$ que poseen invariantes e igualdades de representación calculables y explícitos, y con programas que implementan a las operaciones *opp*, *zero*, *mult* y *dffr* totales sobre los productos de los soportes de las representaciones correspondientes, siendo el dominio del programa que implementa a la operación *suma* el formado por los pares de datos del soporte de la representación para $\mathtt{T}_{cmbn}$ con el mismo grado. De esta forma podremos asignar dominios de definición a los códigos almacenados en la implementación canónica del álgebra que recoge a todas las implementaciones del TAD $\mathcal{T}_{\mathtt{CC}}$ de la categoría anterior.

Cada implementación de la categoría anterior puede verse como una estructura

$$\mathcal{I} = \left(\underline{\mathcal{R}}, (\mathtt{p}^{\mathcal{I}}_{\sigma})_{\sigma \in \{suma, opp, zero, mult, dffr\}}, (\mathtt{p}^{\mathcal{I}}_{invar_{int}}, \mathtt{p}^{\mathcal{I}}_{invar_{cmb}}), (\mathtt{p}^{\mathcal{I}}_{igual_{int}}, \mathtt{p}^{\mathcal{I}}_{igual_{cmbn}})\right)$$

siendo, para $g \in \{int, cmbn\}$, $\mathtt{p}^{\mathcal{I}}_{invar_g}$ un programa que implementa al invariante de la representación $\mathcal{R}_g$ y $\mathtt{p}^{\mathcal{I}}_{igual_g}$ un programa que implementa a la igualdad de la representación $\mathcal{R}_g$.

En este caso, la signatura que modela el trabajo con familias de implementaciones de $\mathcal{T}_{\mathtt{CC}}$ recogiendo parte de la representación es la signatura $\mathtt{CC}_{imp_{rep}}$ con géneros *int*, *cmbn*, *bool* e

$imp_{\mathtt{CC}}$ y las siguientes operaciones

$$\begin{array}{lcllll}
imp_suma & : & imp_{\mathtt{CC}} \ cmbn \ cmbn & \to & cmbn \\
imp_opp & : & imp_{\mathtt{CC}} \ cmbn & \to & cmbn \\
imp_zero & : & imp_{\mathtt{CC}} \ int & \to & cmbn \\
imp_mult & : & imp_{\mathtt{CC}} \ int \ cmbn & \to & cmbn \\
imp_dffr & : & imp_{\mathtt{CC}} \ cmbn & \to & cmbn \\
rep_invar_{int} & : & imp_{\mathtt{CC}} \ int & \to & bool \\
rep_invar_{cmbn} & : & imp_{\mathtt{CC}} \ cmbn & \to & bool \\
rep_igual_{int} & : & imp_{\mathtt{CC}} \ int \ int & \to & bool \\
rep_igual_{cmbn} & : & imp_{\mathtt{CC}} \ cmbn \ cmbn & \to & bool
\end{array}$$

En lo que sigue, sin pérdida de generalidad, fijamos $\mathcal{U}$ como tipo para el género *bool*, con la igualdad que identifica entre sí todos los objetos que no son `eq` con el objeto `nil`.

Una descripción de la $\mathtt{CC}_{imp_{rep}}$-álgebra $\mathcal{M}^{rep,\underline{\mathtt{T}}}$ que recoge a las implementaciones de complejos de cadenas de la categoría $Imp^{\mathtt{T}^{nat}}_{Dec}(\mathcal{T}_{\mathtt{CC}})$, tiene como soportes para los géneros *int*, *cmbn* y *bool* los tipos fijados `integer`, $\mathtt{T}_{cmbn}$ y $\mathcal{U}$, y para el género $imp_{\mathtt{CC}}$, el siguiente conjunto

$$\begin{array}{rl}
\mathcal{M}^{rep,\underline{\mathtt{T}}}_{imp_{\mathtt{CC}}} = \Big\{ & (\mathtt{f}_{invar_{int}}, \ \mathtt{f}_{invar_{cmbn}}, \ \mathtt{f}_{igual_{int}}, \ \mathtt{f}_{igual_{cmbn}}, \ \mathtt{f}_{suma}, \ \mathtt{f}_{opp}, \mathtt{f}_{zero}, \ \mathtt{f}_{mult}, \ \mathtt{f}_{dffr}) \quad | \\
& \exists \ \mathcal{I} = \big(\underline{\mathcal{R}}, \ (\mathtt{p}^{\mathcal{I}}_{\sigma})_{\sigma\in\{suma,opp,zero,mult,dffr\}}, \ (\mathtt{p}^{\mathcal{I}}_{invar_{int}}, \ \mathtt{p}^{\mathcal{I}}_{invar_{cmbn}}), \ (\mathtt{p}^{\mathcal{I}}_{igual_{cmbn}}, \\
& \mathtt{p}^{\mathcal{I}}_{igual_{cmbn}})\big) \ \in \ Obj(Imp^{\mathtt{T}^{nat}}_{Dec}(\mathcal{T}_{\mathtt{CC}})) \text{ tal que } \mathtt{f}_{\sigma} = F(\mathtt{p}^{\mathcal{I}}_{\sigma}) \text{ para cada } \sigma \in \\
& \{suma, opp, zero, mult, dffr\} \text{ y para cada } g \in \{int, cmbn\} \text{ se tiene que } \mathtt{f}_{invar_g} = \\
& F(\mathtt{p}^{\mathcal{I}}_{invar_g}) \text{ y } \mathtt{f}_{igual_g} = F(\mathtt{p}^{\mathcal{I}}_{igual_g}) \ \Big\}
\end{array}$$

Las operaciones vienen definidas por las proyecciones en las componentes correspondientes. En este ejemplo concreto, las condiciones sobre los dominios hacen que éstas vengan dadas como sigue

- La función $rep\_invar_{int_{\mathcal{M}^{rep,\underline{\mathtt{T}}}}}\colon \mathcal{M}^{rep,\underline{\mathtt{T}}}_{imp_{\mathtt{CC}}} \times \mathtt{integer} \to \mathcal{U}$ es total y viene definida por $rep\_invar_{int_{rep,\mathcal{M}^{\underline{\mathtt{T}}}}}(\overline{\mathtt{f}}, \mathtt{i}) := \mathtt{f}_{invar_{int}}(\mathtt{i})$.

- La función parcial $rep\_igual_{int_{\mathcal{M}^{rep,\underline{\mathtt{T}}}}}\colon \mathcal{M}^{rep,\underline{\mathtt{T}}}_{imp_{\mathtt{CC}}} \times \mathtt{integer} \times \mathtt{integer} \to \mathcal{U}$ tiene por dominio $\big\{(\overline{\mathtt{f}}, \mathtt{i}, \mathtt{j}) \ | \ \mathtt{f}_{invar_{int}}(\mathtt{i}) \neq_{\mathtt{eq}} \mathtt{nil} \text{ y } \mathtt{f}_{invar_{int}}(\mathtt{j}) \neq_{\mathtt{eq}} \mathtt{nil}\big\}$ y se define por $rep\_igual_{int_{\mathcal{M}^{rep,\underline{\mathtt{T}}}}}(\overline{\mathtt{f}}, \mathtt{i}, \mathtt{j}) := \mathtt{f}_{igual_{int}}(\mathtt{i}, \mathtt{j})$.

· La función $rep\_invar_{cmbn_{\mathcal{M}^{rep,\underline{\mathtt{T}}}}}\colon \mathcal{M}^{rep,\underline{\mathtt{T}}}_{imp_{\mathtt{CC}}} \times \mathtt{T}_{cmbn} \rightarrow \mathcal{U}$ es total y se define como $rep\_invar_{cmbn_{\mathcal{M}^{rep,\underline{\mathtt{T}}}}}(\overline{\mathtt{f}}, \mathtt{c}) := \mathtt{f}_{invar_{cmbn}}(\mathtt{c})$.

· El dominio de la función $rep\_igual_{cmbn_{\mathcal{M}^{rep,\underline{\mathtt{T}}}}}\colon \mathcal{M}^{rep,\underline{\mathtt{T}}}_{imp_{\mathtt{CC}}} \times \mathtt{T}_{cmbn} \times \mathtt{T}_{cmbn} \rightarrow \mathcal{U}$ es el conjunto $\{(\overline{\mathtt{f}}, \mathtt{c1}, \mathtt{c2}) \mid \mathtt{f}_{invar_{cmbn}}(\mathtt{c1}) \neq_{\mathtt{eq}} \mathtt{nil}$ y $\mathtt{f}_{invar_{cmbn}}(\mathtt{c2}) \neq_{\mathtt{eq}} \mathtt{nil}\}$ y la función viene dada por $rep\_igual_{cmbn_{\mathcal{M}^{rep,\underline{\mathtt{T}}}}}(\overline{\mathtt{f}}, \mathtt{c1}, \mathtt{c2}) := \mathtt{f}_{igual_{cmbn}}(\mathtt{c1}, \mathtt{c2})$.

· La función parcial $imp\_suma_{\mathcal{M}^{rep,\underline{\mathtt{T}}}}\colon \mathcal{M}^{rep,\underline{\mathtt{T}}}_{imp_{\mathtt{CC}}} \times \mathtt{T}_{cmbn} \times \mathtt{T}_{cmbn} \rightarrow \mathtt{T}_{cmbn}$ tiene por dominio $\{(\overline{\mathtt{f}}, \mathtt{c1}, \mathtt{c2}) \mid \mathtt{f}_{invar_{cmbn}}(\mathtt{c1}) \neq_{\mathtt{eq}} \mathtt{nil}$, $\mathtt{f}_{invar_{cmbn}}(\mathtt{c2}) \neq_{\mathtt{eq}} \mathtt{nil}$ y `(= (cmb-dgr c1) (cmb-dgr c2))`$\}$ y viene definida por $imp\_suma_{\mathcal{M}^{rep,\underline{\mathtt{T}}}}(\overline{\mathtt{f}}, \mathtt{c1}, \mathtt{c2}) := \mathtt{f}_{suma}(\mathtt{c1}, \mathtt{c2})$.

· La función $imp\_opp_{\mathcal{M}^{rep,\underline{\mathtt{T}}}}\colon \mathcal{M}^{rep,\underline{\mathtt{T}}}_{imp_{\mathtt{CC}}} \times \mathtt{T}_{cmbn} \rightarrow \mathtt{T}_{cmbn}$ es parcial y tiene por dominio al conjunto $\{(\overline{\mathtt{f}}, \mathtt{c}) \mid \mathtt{f}_{invar_{cmbn}}(\mathtt{c}) \neq_{\mathtt{eq}} \mathtt{nil}\}$, estando definida como $imp\_opp_{\mathcal{M}^{rep,\underline{\mathtt{T}}}}(\overline{\mathtt{f}}, \mathtt{c}) := \mathtt{f}_{opp}(\mathtt{c})$.

· La función $imp\_zero_{\mathcal{M}^{rep,\underline{\mathtt{T}}}}\colon \mathcal{M}^{rep,\underline{\mathtt{T}}}_{imp_{\mathtt{CC}}} \times \mathtt{integer} \rightarrow \mathtt{T}_{cmbn}$ es total y viene definida por $imp\_zero_{\mathcal{M}^{rep,\underline{\mathtt{T}}}}(\overline{\mathtt{f}}, \mathtt{p}) := \mathtt{f}_{zero}(\mathtt{p})$.

· La función $imp\_mult_{\mathcal{M}^{rep,\underline{\mathtt{T}}}}\colon \mathcal{M}^{rep,\underline{\mathtt{T}}}_{imp_{\mathtt{CC}}} \times \mathtt{integer} \times \mathtt{T}_{cmbn} \rightarrow \mathtt{T}_{cmbn}$ es parcial con dominio $\{(\overline{\mathtt{f}}, \mathtt{i}, \mathtt{c}) \mid \mathtt{f}_{invar_{int}}(\mathtt{p}) \neq_{\mathtt{eq}} \mathtt{nil}$ y $\mathtt{f}_{invar_{cmbn}}(\mathtt{c}) \neq_{\mathtt{eq}} \mathtt{nil}\}$ y viene definida por $imp\_mult_{\mathcal{M}^{rep,\underline{\mathtt{T}}}}(\overline{\mathtt{f}}, \mathtt{i}, \mathtt{c}) := \mathtt{f}_{mult}(\mathtt{i}, \mathtt{c})$.

· Finalmente, la función parcial $imp\_dffr_{\mathcal{M}^{rep,\underline{\mathtt{T}}}}\colon \mathcal{M}^{rep,\underline{\mathtt{T}}}_{imp_{\mathtt{CC}}} \times \mathtt{T}_{cmbn} \rightarrow \mathtt{T}_{cmbn}$ tiene por dominio al conjunto $\{(\overline{\mathtt{f}}, \mathtt{c}) \mid \mathtt{f}_{invar_{cmbn}}(\mathtt{c}) \neq_{\mathtt{eq}} \mathtt{nil}\}$ y se define como $imp\_dffr_{\mathcal{M}^{rep,\underline{\mathtt{T}}}}(\overline{\mathtt{f}}, \mathtt{c}) := \mathtt{f}_{dffr}(\mathtt{c})$.

Observar que para definir el dominio de la operación $imp\_zero_{\mathcal{M}^{rep,\underline{\mathtt{T}}}}$ no se utiliza el invariante correspondiente a la representación del género *int*, ya que el papel del género en este caso es distinto que en el caso de la operación $imp\_mult_{\mathcal{M}^{rep,\underline{\mathtt{T}}}}$. En el caso de la primera, se refiere a la graduación y al estar con representaciones naturales debemos considerar todo `integer`.

La implementación canónica para este álgebra, implementación que denotamos por $\mathcal{I}^{rep,\underline{\mathtt{T}},can}$ viene dada como sigue:

- Para los géneros *int*, *cmbn* y *bool* se toman las representaciones literales sobre los tipos `integer`, $\mathtt{T}_{cmbn}$ y $\mathcal{U}$ (con igualdad la interpretación booleana que hacen los intérpretes Lisp), respectivamente.

- Para el género $imp_{\mathtt{CC}}$, la representación $\mathcal{R}^{rep,\underline{\mathtt{T}},can}_{imp_{\mathtt{CC}}}$, que apoyándose en el tipo

```
(defstruct REP-IMP-CC rep-invar-int rep-invar-cmbn rep-igual-int
                      rep-igual-cmbn (:include IMP-CC))
```

  posee como dominio aquellos datos que satisfacen el predicado `D-rep-can-CC-p` que decide si un dato es del tipo anterior y sus campos son objetos funcionales. Este predicado viene

dado del siguiente modo

```
(defun D-rep-can-CC-p (x)
     (and (typep x 'REP-IMP-CC)
          (functionp (REP-IMP-CC-rep-invar-int x))
          (functionp (REP-IMP-CC-rep-invar-cmbn x))
          (functionp (REP-IMP-CC-rep-igual-int x))
          (functionp (REP-IMP-CC-rep-igual-cmbn x))
          (functionp (REP-IMP-CC-imp-suma x))
          (functionp (REP-IMP-CC-imp-opp x))
          (functionp (REP-IMP-CC-imp-zero x))
          (functionp (REP-IMP-CC-imp-mult x))
          (functionp (REP-IMP-CC-imp-dffr x))))
```

Como soporte, consideramos el siguiente conjunto

$\mathcal{S}^{rep,\underline{\mathtt{T}},can}_{imp_{\mathtt{CC}}} = \Big\{ \mathtt{x} \in \mathtt{(satisfies\ D\text{-}rep\text{-}can\text{-}CC\text{-}p)} \mid \exists\, \mathcal{I} = \big(\underline{\mathcal{R}},\ (\mathtt{p}^{\mathcal{I}}_{\sigma})_{\sigma\in\{suma,opp,zero,mult,dffr\}},\ (\mathtt{p}^{\mathcal{I}}_{invar_g})_{int,cmbn},\ (\mathtt{p}^{\mathcal{I}}_{igual_g})_{int,cmbn}\big) \in Obj(Imp^{\mathtt{T}^{nat}}_{Dec}(\mathcal{T}_{\mathtt{CC}}))$ tal que si

$\mathcal{S}^{\mathtt{x}}_{int} = \big\{ \mathtt{d} \in \mathtt{integer} \mid \mathtt{(not\ (eq\ (funcall}\ \mathtt{c}^{\mathcal{I}}_{invar_{int}}\ \mathtt{d)\ nil))} \big\}$ y

$\mathcal{S}^{\mathtt{x}}_{cmbn} = \big\{ \mathtt{c} \in \mathtt{T}_{cmbn} \mid \mathtt{(not\ (eq\ (funcall}\ \mathtt{c}^{\mathcal{I}}_{invar_{cmbn}}\ \mathtt{c)\ nil))} \big\}$

se tiene lo siguiente:

- el $\Omega_{\mathtt{CC}}$-conjunto de la implementación $\mathcal{I}$ es $\big(\big\{(\mathtt{c1},\mathtt{c2}) \in \mathcal{S}^{\mathtt{x}}_{cmbn} \times \mathcal{S}^{\mathtt{x}}_{cmbn} \mid \mathtt{(=\ (cmb\text{-}dgr\ c1)\ (cmb\text{-}dgr\ c2))}\big\}, \mathcal{S}^{\mathtt{x}}_{cmbn}, \mathtt{integer}, \mathcal{S}^{\mathtt{x}}_{int} \times \mathcal{S}^{\mathtt{x}}_{cmbn}, \mathcal{S}^{\mathtt{x}}_{cmbn}\big)$
- $\mathtt{(eq\ c}^{\mathtt{p}^{\mathcal{I}}_{\sigma}}\ \mathtt{(REP\text{-}IMP\text{-}CC\text{-}imp\text{-}}\sigma\ \mathtt{x))}$ para todo $\sigma \in \{suma, opp, zero, mult, dffr\}$
- $\mathtt{(eq\ c}^{\mathtt{p}^{\mathcal{I}}_{invar_g}}\ \mathtt{(REP\text{-}IMP\text{-}CC\text{-}rep\text{-}invar\text{-}g\ x))}$ para $g \in \{int, cmbn\}$
- $\mathtt{(eq\ c}^{\mathtt{p}^{\mathcal{I}}_{igual_g}}\ \mathtt{(REP\text{-}IMP\text{-}CC\text{-}rep\text{-}igual\text{-}g\ x))}$ para $g \in \{int, cmbn\}$ $\Big\}$

La función de abstracción $\alpha^{rep,\underline{\mathtt{T}},can}_{imp_{\mathtt{CC}}} \colon \mathcal{S}^{rep,\underline{\mathtt{T}},can}_{imp_{\mathtt{CC}}} \to \mathcal{M}^{rep,\underline{\mathtt{T}}}_{imp_{\mathtt{CC}}}$ lleva cada registro $\mathtt{x}$ de códigos funcionales a la tupla de funciones definidas por los códigos, con dominios $\big($`integer`, $\mathtt{T}_{cmbn}$, $\mathcal{S}^{\mathtt{x}}_{int} \times \mathcal{S}^{\mathtt{x}}_{int}$, $\mathcal{S}^{\mathtt{x}}_{cmbn} \times \mathcal{S}^{\mathtt{x}}_{cmbn}$, $\big\{(\mathtt{c1},\mathtt{c2}) \in \mathcal{S}^{\mathtt{x}}_{cmbn} \times \mathcal{S}^{\mathtt{x}}_{cmbn} \mid \mathtt{(=\ (cmb\text{-}dgr\ c1)\ (cmb\text{-}dgr\ c2))}\big\}$, $\mathcal{S}^{\mathtt{x}}_{cmbn}$, `integer`, $\mathcal{S}^{\mathtt{x}}_{int} \times \mathcal{S}^{\mathtt{x}}_{cmbn}$, $\mathcal{S}^{\mathtt{x}}_{cmbn}$ $\big)$, respectivamente para los distintos objetos funcionales almacenados en los campos de $\mathtt{x}$.

Como igualdad de representación tomamos la de la abstracción.

- Como programas que implementan a las operaciones, consideramos los siguientes

· $\mathtt{p}^{rep,\underline{\mathtt{T}},can}_{rep\_invar_{int}}$ con soporte $\mathcal{S}^{rep,\underline{\mathtt{T}},can}_{imp_{\mathtt{CC}}} \times \mathtt{integer}$ y código el siguiente objeto funcional

$$\mathtt{c}^{\mathtt{p}^{rep,\underline{\mathtt{T}},can}_{rep\_invar_{int}}} \equiv \mathtt{\#'(lambda\ (x\ i)\ (funcall\ (REP\text{-}IMP\text{-}CC\text{-}rep\text{-}invar\text{-}int\ x)\ i))}$$

A partir del programa anterior definimos, para cada $\mathtt{x} \in \mathcal{S}^{rep,\underline{\mathtt{T}},can}_{imp_{\mathtt{CC}}}$, el conjunto

$$\mathcal{S}^{\mathtt{x}}_{int} = \left\{ \mathtt{i} \in \mathtt{integer} \ \mid \ F(\mathtt{p}^{rep,\underline{\mathtt{T}},can}_{rep\_invar_{int}})(\mathtt{x},\mathtt{i}) \neq_{\mathtt{eq}} \mathtt{nil} \right\}$$

que usaremos para determinar los dominios de otros programas de esta implementación.

· Análogamente, $\mathtt{p}^{rep,\underline{\mathtt{T}},can}_{rep\_invar_{cmbn}}$ tiene por soporte $\mathcal{S}^{rep,\underline{\mathtt{T}},can}_{imp_{\mathtt{CC}}} \times \mathtt{T}_{cmbn}$ y por código el siguiente objeto funcional

$$\mathtt{c}^{\mathtt{p}^{rep,\underline{\mathtt{T}},can}_{rep\_invar_{cmbn}}} \equiv \mathtt{\#'(lambda\ (x\ c)\ (funcall\ (REP\text{-}IMP\text{-}CC\text{-}rep\text{-}invar\text{-}cmbn\ x)\ c))}$$

Análogamente al caso anterior definimos, para cada $\mathtt{x} \in \mathcal{S}^{rep,\underline{\mathtt{T}},can}_{imp_{\mathtt{CC}}}$, el conjunto

$$\mathcal{S}^{\mathtt{x}}_{cmbn} = \left\{ \mathtt{c} \in \mathtt{T}_{cmbn} \ \mid F(\mathtt{p}^{rep,\underline{\mathtt{T}},can}_{rep\_invar_{cmbn}})(\mathtt{x},\mathtt{c}) \neq_{\mathtt{eq}} \mathtt{nil} \right\}$$

· Para cada $(\delta\colon imp_{\mathtt{CC}}\ \omega \to g) \in \{rep\_igual_{int}\ ,\ rep\_igual_{cmbn},\ imp\_opp,\ imp\_zero,\ imp\_mult,\ imp\_dffr\}$, el programa $\mathtt{p}^{rep,\underline{\mathtt{T}},can}_{\delta}$ tiene por soporte al conjunto $\left\{(\mathtt{x},\mathtt{d}_\omega) \in \mathcal{S}^{rep,\underline{\mathtt{T}},can}_{imp_{\mathtt{CC}}} \times \mathtt{T}_\omega \ \mid \ \mathtt{d}_\omega \in \mathcal{S}^{\mathtt{x}}_{\omega}\right\}$ y por código, el siguiente objeto funcional

$$\mathtt{c}^{\mathtt{p}^{rep,\underline{\mathtt{T}},can}_{\delta}} \equiv \mathtt{\#'(lambda\ (x\ d}_\omega\mathtt{)\ (funcall\ (REP\text{-}IMP\text{-}CC\text{-}}\delta\ \mathtt{x)\ d}_\omega\mathtt{))}$$

donde como es natural estamos empleando el subíndice ω en tipos o en conjuntos para denotar al producto, de tipos o de conjuntos respectivamente, de aridad ω.

· El programa $\mathtt{p}^{rep,\underline{\mathtt{T}},can}_{imp\_suma}$ tiene por soporte $\left\{(\mathtt{x},\mathtt{c1},\mathtt{c2}) \in \mathcal{S}^{rep,\underline{\mathtt{T}},can}_{imp_{\mathtt{CC}}} \times \mathcal{S}^{\mathtt{x}}_{cmbn} \times \mathcal{S}^{\mathtt{x}}_{cmbn} \ \mid \ \mathtt{(= (cmb\text{-}dgr\ c1)\ (cmb\text{-}dgr\ c2))}\right\}$ y por código el siguiente objeto funcional

$$\mathtt{c}^{\mathtt{p}^{rep,\underline{\mathtt{T}},can}_{imp\_suma}} \equiv \mathtt{\#'(lambda\ (x\ c1\ c2)\ (funcall\ (REP\text{-}IMP\text{-}CC\text{-}imp\text{-}suma\ x)\ c1\ c2))}$$

La implementación $\mathcal{I}^{rep,\underline{\mathtt{T}},can}$ es objeto final en una subcategoría adecuada de la categoría de implementaciones de $\mathcal{M}^{rep,\underline{\mathtt{T}}}$.

Guiados por lo ya hecho en el capítulo anterior en el que vimos que, bajo determinadas condiciones es posible reducir la expresión del álgebra que recoge a todas las implementaciones de una determinada categoría, vamos ahora a hacer lo análogo en este caso.

Así, para cada implementación $\mathcal{I} = \left(\underline{\mathcal{R}},\ (\mathtt{p}^{\mathcal{I}}_{\sigma})_{\sigma\in\{suma,opp,zero,mult,dffr\}},\ (\mathtt{p}^{\mathcal{I}}_{invar_g})_{g\in\{int,cmbn\}},\ (\mathtt{p}^{\mathcal{I}}_{igual_g})_{g\in\{int,cmbn\}}\right)$ de la categoría $Imp^{\underline{\mathtt{T}}^{nat}}_{Dec}(\mathcal{T}_{\mathtt{CC}})$, denotamos por $\mathcal{S}^{\mathcal{I}}_{int}$ y $\mathcal{S}^{\mathcal{I}}_{cmbn}$ a los soportes de las representaciones para los géneros *int* y *cmbn*, respectivamente, soportes que coinciden con los conjuntos $\left\{\mathtt{i} \in \mathtt{integer} \ \mid \ F(\mathtt{p}^{\mathcal{I}}_{invar_{int}})(\mathtt{i}) \neq_{\mathtt{eq}} \mathtt{nil}\right\}$ y $\left\{\mathtt{c} \in \mathtt{T}_{cmbn} \ \mid \ F(\mathtt{p}^{\mathcal{I}}_{invar_{cmbn}})(\mathtt{c}) \neq_{\mathtt{eq}} \mathtt{nil}\right\}$, respectivamente. Las funciones definidas por algunos de los programas de la implementación anterior tienen determinado su comportamiento. Por una parte, al estar considerando representaciones naturales, el soporte de la representación del género *int* debe ser to-

do `integer` ya que el operador *mult* permite multiplicar por cualquier entero. Así la función $F(\mathtt{p}^{\mathcal{I}}_{invar_{int}})$: `integer` $\to \mathcal{U}$ es total y con comportamiento $F(\mathtt{p}^{\mathcal{I}}_{invar_{int}})(\mathtt{i}) := \mathtt{t}$ para todo $\mathtt{i} \in$ `integer`. Además, por el mismo motivo, la igualdad de la representación debe coincidir con la propia del tipo `integer`, por lo que la función $F(\mathtt{p}^{\mathcal{I}}_{igual_{int}})$: `integer` $\times$ `integer` $\to \mathcal{U}$ es total y viene definida por $F(\mathtt{p}^{\mathcal{I}}_{igual_{int}})(\mathtt{i},\mathtt{j}) :=$ `(= i j)`.

Ahora bien, si recordamos lo visto en el caso de los complejos de cadenas en el capítulo anterior, vemos que es posible definir los operadores de los complejos de cadenas únicamente sobre los generadores y extender por linealidad a las combinaciones. Esto mismo es válido también para el invariante de la representación del género *cmbn* que determina qué combinaciones forman parte del complejo de cadenas que se está representando. Pero además vamos a enriquecer un poco más la construcción que tenemos hasta el momento. Hasta ahora no hemos hecho ninguna referencia especial a la igualdad definida sobre las combinaciones. Sin embargo, si pensamos en Matemáticas, es habitual que sobre un mismo conjunto subyacente, tomando distintas relaciones de equivalencia, representemos distintos conjuntos con igualdad. Así, podríamos partir de un conjunto graduado $(G, =_G)$ en las Matemáticas con una relación de equivalencia (en cada grado). La representación natural del conjunto (con igualdad) anterior poseerá como igualdad de representación la inducida por $=_G$, ya que una representación natural supone un mero cambio del marco de referencia. Realmente, cada conjunto graduado $(\mathtt{G}, =_{\mathtt{G}})$ es un PER graduado. Para cada $p \in \mathbb{Z}$, $(\mathtt{G_p}, =_{\mathtt{G_p}})$ es un par de funciones $(\mathcal{U} \to \mathcal{U}, \mathcal{U} \times \mathcal{U} \to \mathcal{U})$, siendo la primera de ellas total y la segunda parcial con dominio los pares de elementos de $\mathcal{U}$ que no están en la preimagen de `nil` por la primera función.

Así, una forma de codificar cada conjunto graduado `G` en $\mathcal{U}$ es por medio de una función total $\mathtt{invar_G}$: `integer` $\times \mathcal{U} \to \mathcal{U}$ que determina los generadores en cada grado, de forma que $\mathtt{invar_G(p,g)} \neq_{\mathtt{eq}} \mathtt{nil}$ si y solo si $\mathtt{g} \in \mathtt{G_p}$. Una forma de codificar $=_{\mathtt{G}}$ es a través de una función $\mathtt{igual_G}$: `integer` $\times \mathcal{U} \times \mathcal{U} \to \mathcal{U}$ parcial con dominio $\{(\mathtt{p},\mathtt{g1},\mathtt{g2}) \mid \mathtt{invar_G(p,g1)} \neq_{\mathtt{eq}} \mathtt{nil}$ e $\mathtt{invar_G(p,g2)} \neq_{\mathtt{eq}} \mathtt{nil}\}$, y con el siguiente comportamiento $\mathtt{igual_G(p,g1,g2)} := \mathtt{g1} =_{\mathtt{G_p}} \mathtt{g2}$.

Así, si $(\mathtt{G}, =_{\mathtt{G}})$ es el PER graduado que define los generadores del complejo de cadenas implementado por $\mathcal{I}$ (ya considerados en Common Lisp y trasladados de los que viven en las Matemáticas por la representación natural), las funciones definidas por los programas que implementan al invariante y a la igualdad de la representación para el género *cmbn*, $F(\mathtt{p}^{\mathcal{I}}_{invar_{cmbn}})$ y $F(\mathtt{p}^{\mathcal{I}}_{igual_{cmbn}})$, respectivamente, pueden definirse a partir de las funciones que implementan el PER graduado $(\mathtt{G}, =_{\mathtt{G}})$. Así, si $\mathtt{invar}^{\mathcal{I}}_{gen}$: `integer` $\times \mathcal{U} \to \mathcal{U}$ es una función que determina los generadores del complejo de cadenas que se está implementando e $\mathtt{igual}^{\mathcal{I}}_{gen}$: `integer` $\times \mathcal{U} \times \mathcal{U} \to \mathcal{U}$ define la relación de equivalencia $=_{\mathtt{G}}$ entre ellos, es decir, es una función con dominio $\{(\mathtt{p},\mathtt{g1},\mathtt{g2}) \mid \mathtt{invar}^{\mathcal{I}}_{gen}\mathtt{(p,g1)} \neq_{\mathtt{eq}} \mathtt{nil}$ e $\mathtt{invar}^{\mathcal{I}}_{gen}\mathtt{(p,g2)} \neq_{\mathtt{eq}} \mathtt{nil}\}$, y cuyo comportamiento viene descrito como sigue $\mathtt{igual}^{\mathcal{I}}_{gen}\mathtt{(p,g1,g2)} := \mathtt{g1} =_{\mathtt{G_p}} \mathtt{g2}$, las funciones definidas por los programas que implementan al invariante y a la igualdad de la representación del género *cmbn* pueden definirse a partir de ellas del modo descrito a continuación:

- $F(\mathtt{p}^{\mathcal{I}}_{\mathtt{invar}_{cmbn}})$: $\mathtt{T}_{cmbn} \to \mathcal{U}$ es la función total con comportamiento

$$F(\mathtt{p}^{\mathcal{I}}_{\mathtt{invar}_{cmbn}})\big(\langle \mathtt{p}, [(\mathtt{t_1},\mathtt{x_1}),\ldots,(\mathtt{t_m},\mathtt{x_m})]\rangle\big) \neq_{\mathtt{eq}} \mathtt{nil} \text{ si y solo si se verifican}$$

i) $\mathtt{invar}^{\mathcal{I}}_{gen}\mathtt{(p,x_i)} \neq_{\mathtt{eq}} \mathtt{nil}$, para todo $\mathtt{i} \in \{1,\ldots,m\}$

ii) $\mathtt{igual}^{\mathcal{I}}_{gen}\mathtt{(p,x_i,x_j)} =_{\mathtt{eq}} \mathtt{nil}$ para todo $\mathtt{i},\mathtt{j} \in \{1,\ldots,m\}$ con $\mathtt{i} \neq \mathtt{j}$

- $F(\mathtt{p}^{\mathcal{I}}_{igual_{cmbn}})\colon \mathtt{T}_{cmbn} \times \mathtt{T}_{cmbn} \to \mathcal{U}$ es la función parcial que tiene por dominio al conjunto $\{(\mathtt{c1},\mathtt{c2}) \in \mathtt{T}_{cmbn} \times \mathtt{T}_{cmbn} \mid \mathtt{invar}^{\mathcal{I}}_{cmbn}(\mathtt{c1}) \neq_{\mathtt{eq}} \mathtt{nil} \text{ e } \mathtt{invar}^{\mathcal{I}}_{cmbn}(\mathtt{c2}) \neq_{\mathtt{eq}} \mathtt{nil}\}$ y viene definida por $\mathtt{igual}^{\mathcal{I}}_{cmbn}(\langle \mathtt{p}, [(\mathtt{t}_1,\mathtt{x}_1),\ldots,(\mathtt{t}_\mathtt{m},\mathtt{x}_\mathtt{m})]\rangle, \langle \mathtt{q}, [(\mathtt{s}_1,\mathtt{y}_1),\ldots,(\mathtt{s}_\mathtt{n},\mathtt{y}_\mathtt{n})]\rangle) \neq_{\mathtt{eq}} \mathtt{nil}$ si y solo si se verifican:

  i) `(= p q)`

  ii) `(= n m)`

  iii) para cada $\mathtt{i} \in \{1,\ldots,\mathtt{m}\}$, $\exists|\mathtt{j} \in \{1,\ldots,\mathtt{m}\}$ tal que

  - `(= t`$_\mathtt{i}$ `s`$_\mathtt{j}$`)`
  - $\mathtt{igual}^{\mathcal{I}}_{gen}(\mathtt{p},\mathtt{x}_\mathtt{i},\mathtt{y}_\mathtt{j}) \neq_{\mathtt{eq}} \mathtt{nil}$

Volviendo a los operadores de un complejo de cadenas resulta que por las condiciones de $\mathbb{Z}$-módulo en cada grado, el comportamiento de los programas $\mathtt{p}^{\mathcal{I}}_{opp}$, $\mathtt{p}^{\mathcal{I}}_{zero}$ y $\mathtt{p}^{\mathcal{I}}_{mult}$ está determinado y viene dado por el de las funciones $\mathtt{opp}^{\mathcal{U}}$, $\mathtt{zero}^{\mathcal{U}}$ y $\mathtt{mult}^{\mathcal{U}}$ (definidas en la página 197 y que son operadores universales que recogen el comportamiento de las operaciones anteriores) restringidas a los dominios adecuados que, en este caso, son $\mathcal{S}^{\mathcal{I}}_{cmbn}$ para el programa $\mathtt{p}^{\mathcal{I}}_{opp}$, `integer` para $\mathtt{p}^{\mathcal{I}}_{zero}$ e `integer` $\times\mathcal{S}^{\mathcal{I}}_{cmbn}$ para $\mathtt{p}^{\mathcal{I}}_{mult}$.

Respecto al comportamiento del programa $\mathtt{p}^{\mathcal{I}}_{suma}$, algunos comentarios adicionales son necesarios. La función $\mathtt{suma}^{\mathcal{U}}$ (definida en la página 197, y considerada como operador universal en el marco establecido en el capítulo anterior) opera entre sí los monomios con los mismos generadores, considerando como igualdad entre generadores la igualdad `eq`, hecho natural puesto que en la representación en la máquina de una implementación no quedaba recogida ninguna igualdad entre generadores. En ese caso, cualquier relación de igualdad entre las representaciones de los generadores viene recogida exclusivamente en la función de abstracción de la representación. Desde nuestra nueva posición la situación es distinta puesto que se dispone de una igualdad *explícita* entre generadores, la igualdad de la representación, igualdad que debe tenerse en cuenta en la suma de combinaciones. Así, si $\mathtt{igual}^{\mathcal{I}}_{gen}\colon \mathtt{integer} \times \mathcal{U} \times \mathcal{U} \to \mathcal{U}$ es la función que define la igualdad entre generadores en la implementación $\mathcal{I}$, la suma de combinaciones viene dada por la función $\mathtt{suma}^{\mathcal{U},\mathtt{igual}^{\mathcal{I}}_{gen}}\colon \mathtt{T}_{cmbn} \times \mathtt{T}_{cmbn} \to \mathtt{T}_{cmbn}$, función parcial con dominio $\{(\mathtt{c1},\mathtt{c2}) \in \mathcal{S}^{\mathcal{I}}_{cmbn} \times \mathcal{S}^{\mathcal{I}}_{cmbn} \mid \mathtt{(= (cmb\text{-}dgr\ c1)\ (cmb\text{-}dgr\ c2))}\}$ y que se define "concatenando" los monomios de las dos combinaciones a sumar, sumando entre sí los (coeficientes de los) monomios con el mismo (por $\mathtt{igual}^{\mathcal{I}}_{gen}$) dato en el campo `gnr` (es decir, se operan entre sí los monomios con generadores iguales por $\mathtt{igual}^{\mathcal{I}}_{gen}$), eliminando el monomio obtenido si el coeficiente es nulo.

La función $\mathtt{suma}^{\mathcal{U},\mathtt{igual}^{\mathcal{I}}_{gen}}$, a diferencia de la función $\mathtt{suma}^{\mathcal{U}}$, tiene en cuenta una igualdad no trivial entre generadores y, por tanto, depende de la implementación $\mathcal{I}$, aunque se define de modo canónico como acabamos de hacer. Así, el comportamiento del programa $\mathtt{p}^{\mathcal{I}}_{suma}$ viene dado por el de la función $\mathtt{suma}^{\mathcal{U},\mathtt{igual}^{\mathcal{I}}_{gen}}$.

Esto nos lleva a dar otra descripción de la $\mathtt{CC}_{imp_{rep}}$-álgebra $\mathcal{M}^{rep,\underline{\mathtt{T}}}$, descripción que denotamos por $\mathcal{M}^{rep,\underline{\mathtt{T}},red,gen}$ y en la que se recoge la mínima información imprescindible para recuperar un álgebra definida a partir de una implementación de $Imp^{\mathtt{T}^{nat}}_{Dec}(\mathcal{T}_{\mathtt{CC}})$. La descripción

correspondiente al género $imp_{\mathtt{CC}}$ viene dada como sigue

$$\mathcal{M}^{rep,\underline{\mathtt{T}},red,gen}_{imp_{\mathtt{CC}}} = \Big\{(\mathtt{f}_{invar_{gen}} : \mathtt{integer} \times \mathcal{U} \to \mathcal{U},\ \mathtt{f}_{igual_{gen}} : \mathtt{integer} \times \mathcal{U} \times \mathcal{U} \to \mathcal{U},\ \mathtt{f}_{dffr} : \mathtt{T}_{cmbn} \to \mathtt{T}_{cmbn}) \mid \exists\ \mathcal{I} \in Obj(Imp^{\mathtt{T}^{nat}}_{Dec}(\mathcal{T}_{\mathtt{CC}}))\ \text{ verificando que } \mathtt{f}_{dffr} = F(\mathtt{p}^{\mathcal{I}}_{dffr}),\ \mathtt{f}_{invar_{cmbn}} = F(\mathtt{p}^{\mathcal{I}}_{invar_{cmbn}}) \text{ y } \mathtt{f}_{igual_{cmbn}} = F(\mathtt{p}^{\mathcal{I}}_{igual_{cmbn}})\Big\}$$

siendo $\mathtt{f}_{invar_{cmbn}}$ y $\mathtt{f}_{igual_{cmbn}}$ las extensiones por linealidad de las funciones $\mathtt{f}_{invar_{gen}}$ y $\mathtt{f}_{igual_{gen}}$, respectivamente, dadas tal y como se ha visto anteriormente. Es decir, $\mathtt{f}_{invar_{cmbn}} : \mathtt{T}_{cmbn} \to \mathcal{U}$ es la función total definida por $\mathtt{f}_{invar_{cmbn}}\big(\langle \mathtt{p}, [(\mathtt{t}_1, \mathtt{x}_1), \ldots, (\mathtt{t}_\mathtt{m}, \mathtt{x}_\mathtt{m})]\rangle\big) \neq_{\mathtt{eq}} \mathtt{nil}$ si y solo si se verifican:

i) $\mathtt{f}_{invar_{gen}}(\mathtt{p}, \mathtt{x}_\mathtt{i}) \neq_{\mathtt{eq}} \mathtt{nil}$, para todo $\mathtt{i} \in \{1, \ldots, \mathtt{m}\}$

ii) $\mathtt{f}_{igual_{gen}}(\mathtt{p}, \mathtt{x}_\mathtt{i}, \mathtt{x}_\mathtt{j}) =_{\mathtt{eq}} \mathtt{nil}$, para todo $\mathtt{i}, \mathtt{j} \in \{1, \ldots, \mathtt{m}\}$ con $\mathtt{i} \neq \mathtt{j}$

La función $\mathtt{f}_{igual_{cmbn}} : \mathtt{integer} \times \mathtt{T}_{cmbn} \times \mathtt{T}_{cmbn} \to \mathcal{U}$ es parcial con dominio el conjunto $\big\{(\mathtt{c1}, \mathtt{c2}) \in \mathtt{T}_{cmbn} \times \mathtt{T}_{cmbn} \mid \mathtt{f}_{invar_{cmbn}}(\mathtt{c1}) \neq_{\mathtt{eq}} \mathtt{nil} \text{ y } \mathtt{f}_{invar_{cmbn}}(\mathtt{c2}) \neq_{\mathtt{eq}} \mathtt{nil}\big\}$ y está definida por $\mathtt{f}_{igual_{cmbn}}\big(\langle \mathtt{p}, [(\mathtt{t}_1, \mathtt{x}_1), \ldots, (\mathtt{t}_\mathtt{m}, \mathtt{x}_\mathtt{m})]\rangle, \langle \mathtt{q}, [(\mathtt{s}_1, \mathtt{y}_1), \ldots, (\mathtt{s}_\mathtt{m}, \mathtt{y}_\mathtt{m})]\rangle\big) \neq_{\mathtt{eq}} \mathtt{nil}$ si y solo si se verifican:

i) `(= p q)`

ii) `(= n m)`

iii) para cada $\mathtt{i} \in \{1, \ldots, \mathtt{m}\}$, $\exists | \mathtt{j} \in \{1, \ldots, \mathtt{m}\}$ tal que

- `(= t_i s_j)`
- $\mathtt{f}_{igual_{gen}}(\mathtt{p}, \mathtt{x}_\mathtt{i}, \mathtt{y}_\mathtt{j}) \neq_{\mathtt{eq}} \mathtt{nil}$

En este caso, no todas las operaciones vienen definidas por las proyecciones en las componentes correspondientes. Concretamente, las componentes correspondientes al invariante del género *cmbn* y a la igualdad sobre los datos de ese género en cada $\overline{\mathtt{f}} \in \mathcal{M}^{rep,\underline{\mathtt{T}},red,gen}$, no trabajan sobre cualquier combinación sino exclusivamente sobre generadores, por lo que vienen dadas como sigue:

- $rep\_invar_{cmbn_{\mathcal{M}^{rep,\underline{\mathtt{T}},red,gen}}}$ es la función total definida por $rep\_invar_{cmbn_{\mathcal{M}^{rep,\underline{\mathtt{T}},red,gen}}}(\overline{\mathtt{f}}, \mathtt{c}) := \mathtt{f}_{invar_{cmbn}}(\mathtt{c})$.
- La función $rep\_igual_{cmbn_{\mathcal{M}^{rep,\underline{\mathtt{T}},red,gen}}}$ tiene por dominio $\big\{(\overline{\mathtt{f}}, \mathtt{c1}, \mathtt{c2}) \mid \mathtt{f}_{invar_{cmbn}}(\mathtt{c1}) \neq_{\mathtt{eq}} \mathtt{nil} \text{ y } \mathtt{f}_{invar_{cmbn}}(\mathtt{c2}) \neq_{\mathtt{eq}} \mathtt{nil}\big\}$ definida por $rep\_igual_{cmbn_{\mathcal{M}^{rep,\underline{\mathtt{T}},red,gen}}}(\overline{\mathtt{f}}, \mathtt{c1}, \mathtt{c2}) := \mathtt{f}_{igual_{cmbn}}(\mathtt{c1}, \mathtt{c2})$.

La implementación canónica $\mathcal{I}^{rep,\underline{\mathtt{T}},red,gen,can}$ del modelo anterior viene dada como sigue

- Representaciones literales sobre los tipos `integer`, $\mathtt{T}_{cmbn}$ y $\mathcal{U}$.
- Para definir la representación del género $imp_{\mathtt{CC}}$, nos apoyamos en el siguiente tipo

```
(defstruct REP-IMP-CC-red-gen rep-invar-gen rep-igual-gen imp-dffr)
```

y tomamos como dominio los objetos Lisp que satisfacen el predicado

```
(defun D-rep-can-CC-red-gen-p (x)
     (and (typep x 'REP-IMP-CC-red-gen)
          (functionp (REP-IMP-CC-red-gen-rep-invar-gen x))
          (functionp (REP-IMP-CC-red-gen-rep-igual-gen x))
          (functionp (REP-IMP-CC-red-gen-imp-dffr x))))
```

que decide si un dato es del tipo anterior y sus campos son o no objetos funcionales.

El soporte $\mathcal{S}^{rep,\underline{\mathtt{T}},red,gen,can}_{imp_{\mathtt{CC}}}$ viene definido como el siguiente conjunto

$$\mathcal{S}^{rep,\underline{\mathtt{T}},red,gen,can}_{imp_{\mathtt{CC}}} = \Big\{ \mathtt{x} \in \texttt{(satisfies D-rep-can-CC-red-gen-p)} \mid \exists\ \mathcal{I} = \big(\underline{\mathcal{R}}, (\mathtt{p}^{\mathcal{I}}_{\sigma})_{\sigma\in\{suma,opp,zero,mult,dffr\}}, (\mathtt{p}^{\mathcal{I}}_{invar_g})_{g\in\{int,cmbn\}}, (\mathtt{p}^{\mathcal{I}}_{igual_g})_{g\in\{int,cmbn\}}\big) \in Obj(Imp^{\mathtt{T}^{nat}}_{Dec}(\mathcal{T}_{\mathtt{CC}})) \text{ tal que denotando}$$

$\mathcal{S}^{\mathcal{I}}_{gen} = \{(\mathtt{p},\ \mathtt{g}) \in \texttt{integer} \times \mathcal{U} \mid \texttt{(not (eq (funcall } \mathtt{c}^{\mathcal{I}}_{invar_{cmbn}}$ `(make-cmb :dgr p :lst (list (make-mnm :cff 1 :gnr g)))) nil))` $\}$ y

$\mathcal{S}^{\mathcal{I}}_{cmbn} = \big\{\langle \mathtt{p}, [(\mathtt{t}_1,\mathtt{x}_1),\ldots,(\mathtt{t}_{\mathtt{m}},\mathtt{x}_{\mathtt{m}})]\rangle \in \mathtt{T}_{cmbn} \mid (\mathtt{p},\mathtt{x}_{\mathtt{i}}) \in \mathcal{S}^{\mathcal{I}}_{gen}$ para todo $\mathtt{i} \in \{1,\ldots,\mathtt{m}\}$ y `(eq (funcall` $\mathtt{c}^{\mathcal{I}}_{igual_{cmbn}}$ `p` $\mathtt{x}_{\mathtt{i}}$ $\mathtt{x}_{\mathtt{j}}$`) nil)` para todo $\mathtt{i},\mathtt{j} \in \{1,\ldots,\mathtt{m}\}$ con $\mathtt{i} \neq \mathtt{j}\big\}$

se tiene que

- Existe un programa $\mathtt{p}^{\mathtt{x}}_{invar_{gen}}$ con soporte $\mathcal{S}^{\mathcal{I}}_{gen}$ verificando:
  - *i*) `(eq` $\mathtt{c}^{\mathtt{p}^{\mathtt{x}}_{invar_{gen}}}$ `(REP-IMP-CC-red-gen-rep-invar-gen x))`
  - *ii*) La función definida por el programa $\mathtt{p}^{\mathtt{x}}_{invar_{gen}}$ extendida por linealidad a $\mathtt{T}_{cmbn}$, tal y como se ha extendido en el modelo, coincide con la función definida por el programa $F(\mathtt{p}^{\mathcal{I}}_{invar_{cmbn}})$
- Existe un programa $\mathtt{p}^{\mathtt{x}}_{igual_{gen}}$ con soporte $\big\{(\mathtt{p},\mathtt{g1},\mathtt{g2}) \in \texttt{integer} \times \mathcal{U}\times\mathcal{U} \mid (\mathtt{p},\mathtt{g1}),(\mathtt{p},\mathtt{g2}) \in \mathcal{S}^{\mathcal{I}}_{gen}\big\}$ verificando:
  - *i*) `(eq` $\mathtt{c}^{\mathtt{p}^{\mathtt{x}}_{igual_{gen}}}$ `(REP-IMP-CC-red-gen-rep-igual-gen x))`
  - *ii*) La función definida por el programa $\mathtt{p}^{\mathtt{x}}_{igual_{gen}}$ extendida por linealidad a $\big\{(\mathtt{c1},\mathtt{c2}) \in \mathcal{S}^{\mathcal{I}}_{cmbn} \times \mathcal{S}^{\mathcal{I}}_{cmbn} \mid$ `(= (cmb-dgr c1) (cmb-dgr c2))`$\big\}$ coincide con la función definida por el programa $F(\mathtt{p}^{\mathcal{I}}_{igual_{cmbn}})$
- el $\Omega_{\mathtt{CC}}$-conjunto de $\mathcal{I}$ es $\big(\{(\mathtt{c1},\mathtt{c2}) \in \mathcal{S}^{\mathcal{I}}_{cmbn}\times\mathcal{S}^{\mathcal{I}}_{cmbn} \mid$ `(= (cmb-dgr c1) (cmb-dgr c2))` $\}$, $\mathcal{S}^{\mathcal{I}}_{cmbn}$, `integer`, `integer` $\times\ \mathcal{S}^{\mathcal{I}}_{cmbn}$, $\mathcal{S}^{\mathcal{I}}_{cmbn}\big)$
- `(eq` $\mathtt{c}^{\mathtt{p}^{\mathcal{I}}_{dffr}}$ `(REP-IMP-CC-red-gen-imp-dffr x))` $\Big\}$

La función de abstracción $\alpha^{rep,\underline{\mathtt{T}},red,gen,can}_{imp_{\mathtt{CC}}} : \mathcal{S}^{rep,\underline{\mathtt{T}},red,gen,can}_{imp_{\mathtt{CC}}} \to \mathcal{M}^{rep,\underline{\mathtt{T}},red,gen}_{imp_{\mathtt{CC}}}$ asocia a cada dato $\mathtt{x}$ del soporte la tupla de funciones $\big(\mathtt{f}^{\mathtt{x}}_{invar_{gen}}, \mathtt{f}^{\mathtt{x}}_{igual_{gen}}, \mathtt{f}^{\mathtt{x}}_{dffr}\big)$ definidas por los códigos

almacenados en los campos de $\mathtt{x}$ sobre los dominios $\mathcal{S}^{\mathcal{I}}_{gen}$, $\{(\mathtt{p},\mathtt{g1},\mathtt{g2}) \in \mathtt{integer}\times\mathcal{U}\times\mathcal{U} \mid (\mathtt{p},\mathtt{g1}),(\mathtt{p},\mathtt{g2}) \in \mathcal{S}^{\mathcal{I}}_{gen}\}$ y $\mathcal{S}^{\mathcal{I}}_{cmbn}$, respectivamente.

Como igualdad de representación tomamos la de la abstracción.

- Como programas que implementan a las operaciones, consideramos los siguientes

  · $\mathtt{p}^{rep,\underline{\mathtt{T}},red,gen,can}_{rep\_invar_{int}}$ con soporte $\mathcal{S}^{rep,\underline{\mathtt{T}},red,gen,can}_{imp_{\mathtt{CC}}} \times \mathtt{integer}$ y el siguiente código

  $$\mathtt{c}^{\mathtt{p}^{rep,\underline{\mathtt{T}},red,gen,can}_{rep\_invar_{int}}} \equiv \mathtt{\#'(lambda\ (x\ i)\quad t)}$$

  · $\mathtt{p}^{rep,\underline{\mathtt{T}},red,gen,can}_{rep\_igual_{int}}$ con soporte $\mathcal{S}^{rep,\underline{\mathtt{T}},red,gen,can}_{imp_{\mathtt{CC}}} \times \mathtt{integer} \times \mathtt{integer}$ y código

  $$\mathtt{c}^{\mathtt{p}^{rep,\underline{\mathtt{T}},red,gen,can}_{rep\_igual_{int}}} \equiv \mathtt{\#'(lambda\ (x\ i\ j)\quad (=\ i\ j))}$$

  · $\mathtt{p}^{rep,\underline{\mathtt{T}},red,gen,can}_{rep\_invar_{cmbn}}$ tiene por soporte $\mathcal{S}^{rep,\underline{\mathtt{T}},red,gen,can}_{imp_{\mathtt{CC}}} \times \mathtt{T}_{cmbn}$ y por código el siguiente objeto funcional

  $$\mathtt{c}^{\mathtt{p}^{rep,\underline{\mathtt{T}},can}_{rep\_invar_{cmbn}}} \equiv \mathtt{\#'(lambda\ (x\ c)\ (funcall\ c}^{\mathtt{x}}_{rep\_invar_{cmbn}}\ \mathtt{c))}$$

  siendo $\mathtt{c}^{\mathtt{x}}_{rep\_invar_{cmbn}}$ un objeto funcional que extiende el comportamiento de `(REP-IMP-CC-red-gen-rep-invar-gen x)` a las combinaciones tal y como se ha comentado anteriormente.

  Para cada $\mathtt{x} \in \mathcal{S}^{rep,\underline{\mathtt{T}},red,gen,can}_{imp_{\mathtt{CC}}}$, definimos el siguiente conjunto

  $$\mathcal{S}^{\mathtt{x}}_{cmbn} = \{\mathtt{c} \in \mathtt{T}_{cmbn} \mid F(\mathtt{p}^{rep,\underline{\mathtt{T}},red,gen,can}_{rep\_invar_{cmbn}})(\mathtt{x},\mathtt{c}) \neq_{\mathtt{eq}} \mathtt{nil}\}$$

  conjunto que nos servirá para definir los soportes de otros programas.

  · El programa $\mathtt{p}^{rep,\underline{\mathtt{T}},red,gen,can}_{rep\_igual_{cmbn}}$ tiene por soporte al conjunto $\{(\mathtt{x},\mathtt{c1},\mathtt{c2}) \in \mathcal{S}^{rep,\underline{\mathtt{T}},red,gen,can}_{imp_{\mathtt{CC}}} \times \mathtt{T}_{cmbn} \times \mathtt{T}_{cmbn} \mid \mathtt{c1},\mathtt{c2} \in \mathcal{S}^{\mathtt{x}}_{cmbn}\}$ y por código el siguiente objeto funcional

  $$\mathtt{c}^{\mathtt{p}^{rep,\underline{\mathtt{T}},can}_{rep\_igual_{cmbn}}} \equiv \mathtt{\#'(lambda\ (x\ c1\ c2)\ (funcall\ c}^{\mathtt{x}}_{rep\_igual_{cmbn}}\ \mathtt{c1\ c2))}$$

  siendo $\mathtt{c}^{\mathtt{x}}_{rep\_igual_{cmbn}}$ un objeto funcional que extiende el comportamiento del objeto funcional `(REP-IMP-CC-red-gen-rep-igual-gen x)` que determina el comportamiento únicamente para generadores.

  · Para cada $(\delta\colon imp_{\mathtt{CC}}\ \omega \rightarrow g) \in \{imp\_opp,\ imp\_zero,\ imp\_mult\}$, el programa $\mathtt{p}^{rep,\underline{\mathtt{T}},red,gen,can}_{\delta}$ tiene por soporte $\{(\mathtt{x},\mathtt{d}_\omega) \in \mathcal{S}^{rep,\underline{\mathtt{T}},red,gen,can}_{imp_{\mathtt{CC}}} \times \mathtt{T}_\omega \mid \mathtt{d}_\omega \in \mathcal{S}^{\mathtt{x}}_{\omega}\}$ y por código, el siguiente objeto funcional

  $$\mathtt{c}^{\mathtt{p}^{rep,\underline{\mathtt{T}},red,gen,can}_{\delta}} \equiv \mathtt{\#'(lambda\ (x\ d}_\omega\mathtt{)\ (funcall\ c}^{\mathtt{p}_{\delta^{\mathcal{U}}}}\ \mathtt{d}_\omega\mathtt{))}$$

  · El programa $\mathtt{p}^{rep,\underline{\mathtt{T}},red,gen,can}_{imp\_suma}$ tiene por soporte al conjunto $\{(\mathtt{x},\mathtt{c1},\mathtt{c2}) \in \mathcal{S}^{rep,\underline{\mathtt{T}},red,gen,can}_{imp_{\mathtt{CC}}} \times \mathcal{S}^{\mathtt{x}}_{cmbn} \times \mathcal{S}^{\mathtt{x}}_{cmbn} \mid \mathtt{(=\ (cmb\text{-}dgr\ c1)\ (cmb\text{-}dgr\ c2))}\}$ y por código

el siguiente objeto funcional

$$\mathtt{c}^{\mathtt{p}_{imp\_suma}^{rep,\underline{\mathtt{T}},red,gen,can}} \equiv \mathtt{\#'(lambda\ (x\ c1\ c2)\ (funcall\ c}^{\mathtt{p}_{\mathtt{suma}}^{\mathcal{U},\mathtt{f}^{\mathtt{x}}_{igual}}}\ \mathtt{c1\ c2))}$$

siendo $\mathtt{c}^{\mathtt{p}_{\mathtt{suma}}^{\mathcal{U},\mathtt{f}^{\mathtt{x}}_{igual}}}$ el código de un programa que implemente a la función $\mathtt{suma}^{\mathcal{U},\mathtt{f}^{\mathtt{x}}_{igual}}$ (definida en la página 234), siendo $\mathtt{f}^{\mathtt{x}}_{igual}$ la función con dominio $\{(\mathtt{p},\mathtt{g1},\mathtt{g2}) \in \mathtt{integer} \times \mathcal{U} \times \mathcal{U} \mid (\mathtt{p},\mathtt{g1}),(\mathtt{p},\mathtt{g2}) \in \mathcal{S}^{\mathcal{I}}_{gen}\}$ y definida por el objeto funcional `(REP-IMP-CC-red-gen-rep-igual-gen x)`.

- El programa $\mathtt{p}^{rep,\underline{\mathtt{T}},red,gen,can}_{imp\_dffr}$ tiene por soporte $\{(\mathtt{x},\mathtt{c}) \in \mathcal{S}^{rep,\underline{\mathtt{T}},red,gen,can}_{imp_{\mathtt{CC}}} \times \mathtt{T}_{cmbn} \mid \mathtt{c} \in \mathcal{S}^{\mathtt{x}}_{cmbn}\}$ y por código el siguiente objeto funcional

$$\mathtt{c}^{\mathtt{p}^{rep,\underline{\mathtt{T}},can}_{imp\_dffr}} \equiv \mathtt{\#'(lambda\ (x\ c)\ (funcall\ (REP\text{-}IMP\text{-}CC\text{-}red\text{-}gen\text{-}imp\text{-}dffr\ x)\ c))}$$

Aún es posible dar otra descripción de la $\mathtt{CC}_{imp_{rep}}$-álgebra $\mathcal{M}^{rep,\underline{\mathtt{T}}^{nat}}$ que difiere de esta última únicamente en que las funciones almacenadas asociadas a la diferencial solo están definidas sobre generadores. Tal y como hemos visto en el capítulo anterior, serán funciones con aridad $\mathtt{f}_{\mathit{dffr}_{gen}}\colon \mathtt{integer} \times \mathcal{U} \to \mathtt{T}_{cmbn}$ y definidas sobre los generadores de forma que pueden extenderse por linealidad dando lugar a funciones $\mathtt{f}_{\mathit{dffr}_{cmbn}}\colon \mathtt{T}_{cmbn} \to \mathtt{T}_{cmbn}$ que determinan el comportamiento de la diferencial sobre cualquier combinación del complejo de cadenas del que provienen. La descripción correspondiente al soporte del género $imp_{\mathtt{CC}}$ viene dada en este caso como sigue:

$$\begin{aligned}\mathcal{M}^{rep,\underline{\mathtt{T}},red,gen'}_{imp_{\mathtt{CC}}} = \Big\{&(\mathtt{f}_{invar_{gen}}\colon \mathtt{integer} \times \mathcal{U} \to \mathcal{U},\ \mathtt{f}_{igual_{gen}}\colon \mathtt{integer} \times \mathcal{U} \times \mathcal{U} \to \mathcal{U},\\ &\mathtt{f}_{\mathit{dffr}_{gen}}\colon \mathtt{integer} \times \mathcal{U} \to \mathtt{T}_{cmbn}) \mid \exists\, \mathcal{I} \in Obj(Imp^{\mathtt{T}^{nat}}_{Dec}(\mathcal{T}_{\mathtt{CC}})) \text{ verificando que}\\ &\mathtt{f}_{\mathit{dffr}_{cmbn}} = F(\mathtt{p}^{\mathcal{I}}_{\mathit{dffr}}),\ \mathtt{f}_{invar_{cmbn}} = F(\mathtt{p}^{\mathcal{I}}_{invar_{cmbn}}) \text{ y } \mathtt{f}_{igual_{cmbn}} = F(\mathtt{p}^{\mathcal{I}}_{igual_{cmbn}})\Big\}\end{aligned}$$

siendo $\mathtt{f}_{invar_{cmbn}}$ y $\mathtt{f}_{igual_{cmbn}}$ las extensiones por linealidad de las funciones $\mathtt{f}_{invar_{gen}}$ y $\mathtt{f}_{igual_{gen}}$, respectivamente, y $\mathtt{f}_{\mathit{dffr}_{cmbn}}$ la extensión correspondiente a $\mathtt{f}_{\mathit{dffr}_{gen}}$.

Siguiendo nuestro propósito de acercarnos lo más posible a EAT, decir que ambas posibilidades quedan recogidas en EAT. Junto a cada tupla de códigos funcionales que el programa almacena, recoge también información sobre cuál de las dos estrategias es la que está siguiendo, con el fin de conocer la aridad del operador diferencial del que se dispone. Es decir, almacena información para saber si disponemos del código que trabaja sobre combinaciones o del que trabaja únicamente sobre generadores. Esto corresponde a trabajar con la $\mathtt{CC}_{imp_{rep}}$-álgebra

$$\begin{aligned}\mathcal{M}^{rep,\underline{\mathtt{T}},EAT}_{imp_{\mathtt{CC}}} = \Big\{&(\mathtt{f}_{invar_{gen}}\colon \mathtt{integer} \times \mathcal{U} \to \mathcal{U},\ \mathtt{f}_{igual_{gen}}\colon \mathtt{integer} \times \mathcal{U} \times \mathcal{U} \to \mathcal{U},\ \mathtt{f}_{str}\colon \to\\ &\{:\mathtt{gnr}, :\mathtt{cmb}\},\ \mathtt{f}_{\mathit{dffr}}) \mid \exists\, \mathcal{I} \in Obj(Imp^{\mathtt{T}^{nat}}_{Dec}(\mathcal{T}_{\mathtt{CC}})) \text{ verificando que } \mathtt{f}_{\mathit{dffr}_{cmbn}} =\\ &F(\mathtt{p}^{\mathcal{I}}_{\mathit{dffr}}),\ \mathtt{f}_{invar_{cmbn}} = F(\mathtt{p}^{\mathcal{I}}_{invar_{cmbn}}),\ \mathtt{f}_{igual_{cmbn}} = F(\mathtt{p}^{\mathcal{I}}_{igual_{cmbn}}) \text{ y si } \mathtt{f}_{str}() =_{\mathtt{eq}}\\ &:\mathtt{gnr} \text{ entonces } \mathtt{f}_{\mathit{dffr}}\colon \mathtt{integer} \times \mathcal{U} \to \mathtt{T}_{cmbn} \text{ es la función } \mathtt{f}_{\mathit{dffr}_{cmbn}} = F(\mathtt{p}^{\mathcal{I}}_{\mathit{dffr}});\\ &\text{en caso contrario, } \mathtt{f}_{\mathit{dffr}}\colon \mathtt{T}_{cmbn} \to \mathtt{T}_{cmbn} \text{ es } \mathtt{f}_{\mathit{dffr}} = F(\mathtt{p}^{\mathcal{I}}_{\mathit{dffr}})\Big\}\end{aligned}$$

siendo $\mathtt{f}_{invar_{cmbn}}$, $\mathtt{f}_{igual_{cmbn}}$ y $\mathtt{f}_{\mathit{dffr}_{cmbn}}$ las extensiones por linealidad de las funciones $\mathtt{f}_{invar_{gen}}$, $\mathtt{f}_{igual_{gen}}$ y $\mathtt{f}_{\mathit{dffr}}$, respectivamente.

Otro comentario que consideramos necesario en este punto es que en EAT, por motivos de eficiencia, los invariantes no están recogidos en las estructuras de datos. Esto nos lleva a que si no disponemos de ellos, teóricamente no hay forma de asignar a cada operación un dominio de definición, lo que nos vuelve a llevar a la situación recogida en el capítulo anterior. Ahora bien, este comentario debe entenderse únicamente desde el punto de vista teórico, ya que en la práctica no se requieren explícitamente los dominios de definición de los programas. Además, este último modelo es el que más fielmente recoge el modo de trabajo de EAT puesto que contiene toda la información que EAT utiliza, concretamente, igualdades de representación y diferenciales.

Hasta ahora, las estructuras de datos que hemos utilizado para recoger implementaciones de TADs, por ejemplo de complejos de cadenas, son estructuras en las que cada dato es una tupla de objetos funcionales que provienen de las implementaciones de partida. Esta forma de codificar las estructuras algebraicas, como tuplas de objetos funcionales, permite codificar estructuras de naturaleza infinita. Sin embargo, si nos planteamos el tipo de información que a partir de ellas somos capaces de extraer, nos damos cuenta que solo vamos a poder deducir información local de las estructuras a las que codifican. Por ejemplo, en el caso de los complejos de cadenas, conocemos el comportamiento de los operadores sobre los elementos de los grupos del complejo de cadenas implementado. Sin embargo, no tenemos ninguna información global sobre los grupos (por ejemplo, no sabemos si son o no vacíos y no conocemos su cardinal) puesto que las representaciones funcionales no permiten implementar algunos operadores de naturaleza global, tal y como hemos visto en la sección 3.6.2. Esto está relacionado con el hecho de que la implementación del complejo de cadenas que se está manejando es parcial en el sentido de que no recoge toda la información del complejo de cadenas. Por ejemplo, en el caso de complejos de cadenas sobre grupos finitamente generados, que es un caso en el que EAT trabaja, al definir matemáticamente el complejo de cadenas se tienen explícitamente los generadores y eso es distinto que tener una función que decide si un elemento de $\mathcal{U}$ es o no un generador. En la definición matemática hay más información.

Todo esto nos acerca a lo que, en terminología de Sergeraert, se conoce como *objetos localmente efectivos*. La idea detrás de la característica anterior de un objeto radica en que son objetos cuyo comportamiento local conocemos, pero no tenemos información sobre sus características globales. La representación en la máquina de un complejo de cadenas, tal cual la hemos visto hasta ahora, es un objeto localmente efectivo. Esto tiene que ver con el hecho de que los objetos funcionales que representan en la máquina un complejo de cadenas, definen matemáticamente el complejo de cadenas, sin embargo, la definición no es constructiva y ése es el motivo por el que no se puede obtener conocimiento global.

Frente a la propiedad de objeto localmente efectivo se encuentra la de *objeto efectivo*. Básicamente la propiedad de que un objeto sea efectivo se traduce en la posibilidad de deducir a partir de él información global acerca de la estructura a la que está representando.

En la siguiente sección abordamos la propiedad de efectividad de un objeto.

4.3 Especificaciones, efectividad e infinitud

Vamos a intentar establecer la relación entre las tres nociones del título de la sección a partir de un ejemplo.

Si pensamos en las listas tal y como normalmente se utilizan, podemos considerar la siguiente

signatura

$$\begin{array}{lcccc} nil & : & & \to & l \\ cons & : & elem \;\; l & \to & l \\ first & : & l & \to & elem \\ rest & : & l & \to & l \end{array}$$

en la que entendemos que *nil* y *cons* son los constructores, de forma que las listas son todo lo que se genera a partir de los constructores anteriores, es decir, las listas se obtienen como álgebra libre a partir de la signatura formada por *nil* y *cons*. Los otros dos operadores, $first$ y *rest* son de acceso.

Además de la anterior, hay otras formas de presentar las listas, por ejemplo por medio de la siguiente signatura

$$\begin{array}{lcccc} nil & : & & \to & l \\ cons & : & elem \;\; l & \to & l \\ l\text{-}nth & : & l \;\; nat & \to & elem \\ l\text{-}length & : & l & \to & nat \end{array}$$

En este caso los constructores son también *nil* y *cons*. Las operaciones *l-nth* y *l-length* son operaciones de acceso que pueden definirse a partir de las operaciones de la primera signatura.

Presentamos una tercera forma de trabajar con listas en la línea de lo hecho con nuestros objetos localmente efectivos. Olvidando los constructores, los operadores de acceso darían lugar a una signatura puramente deconstructora, tomando como primitivas las siguientes operaciones

$$\begin{array}{lcccc} nth & : & nat & \to & elem \\ length & : & & \to & nat \end{array}$$

Esta última signatura no recoge cómo construir listas, sin embargo, sus operaciones permiten "recuperar" toda la información asociada a una lista.

Este ejemplo pone de manifiesto la existencia de dos puntos de vista en el marco de especificación de tipos de datos. Por un lado el marco algebraico, basado en la semántica inicial y en el que se trabaja con TADs. Por otro, el marco coalgebraico, marco que se apoya en la semántica final y que es el utlizado en la orientación a objetos. En la literatura existen diversos trabajos en los que la relación entre ambos marcos queda explicada, por ejemplo en [29].

Los dos puntos de vista anteriores no son totalmente equivalentes. El segundo de ellos es más general, en el sentido de que permite especificar objetos infinitos, algo que no es posible en el primero. En el caso particular de las listas, partiendo de la tercera signatura y olvidando la operación *length*, solo con *nth* es posible recuperar todas las secuencias infinitas de *nil* y *cons*. Si además, se permite parcialidad, también se recuperan las listas finitas. Sin embargo, hay

un aspecto en el que el segundo punto de vista es más débil que es que no permite "construir" listas. Algunas de ellas por problemas de cardinalidad (las infinitas) y las otras por problemas de indecidibilidad (las finitas).

Esta debilidad del segundo punto de vista ya es conocida tanto en el marco coalgebraico, como en el de la orientación a objetos, y, en particular, cuando se trabaja con estructuras infinitas.

El enlace entre los dos puntos de vista es la manipulación de estructuras infinitas. Esto es posible en lenguajes de programación con evaluación lazy (como Haskell [14]). Otros lenguajes, como ML o Common Lisp, pese a no poseer evaluación lazy, permiten simularla lo que, en particular se traduce en la posibilidad de manipular estructuras infinitas. En Common Lisp la evaluación lazy puede simularse por medio de los cierres léxicos. La representación en la máquina de una lista infinita como un cierre léxico es un objeto funcional de la forma `#'(lambda (i) ... )` devolviendo un dato del tipo de los elementos de las listas.

Todo esto nos acerca a lo que, en terminología de Sergeraert, se conoce como *objetos localmente efectivos*. La idea detrás de la característica anterior de un objeto radica en que son objetos cuyo comportamiento local conocemos, pero no tenemos información sobre sus características globales. La representación en la máquina de una lista como un cierre léxico `#'(lambda (i) ... )` es un objeto localmente efectivo: nos permite saber qué elemento ocupa cada posición en la lista, sin embargo, a partir del cierre léxico no es posible obtener ninguna información global sobre la lista a la que implementa. Por ejemplo, no sabemos si es o no vacía. Lo anterior tiene su explicación en el hecho de que los objetos funcionales que representan en la máquina una lista definen completamente la lista desde el punto de vista matemático; sin embargo, no permiten construir la lista ni obtener conocimiento global a partir de ellos. La implementación de una lista como un cierre léxico no recoge toda la información de la lista, en ese sentido, podemos pensar que es una implementación parcial.

Frente a los objetos localmente efectivos están los *objetos efectivos*. A grandes rasgos, la propiedad de efectividad de un objeto se traduce en la posibilidad de deducir de él información global acerca de la estructura a la que está representando.

La propiedad de efectividad o no de un objeto está también relacionada con la naturaleza finita o infinita de la estructura matemática a la que representa. Un objeto efectivo recoge más fielmente toda la información de la estructura, pero la estructura a la que representa siempre tiene que ser de naturaleza finita, en el sentido de que sus elementos puedan construirse a partir de un número finito de datos. Sin embargo, un objeto localmente efectivo solo recoge la parte comportamental de la estructura matemática, por lo que en este caso no existe ninguna restricción sobre la naturaleza de la estructura que puede ser finita o infinita.

Así, una estructura de naturaleza finita puede codificarse por medio de un objeto localmente efectivo que únicamente recoge el comportamiento de sus operaciones, o por un objeto efectivo que, además de lo anterior, contendrá información que permite acceder a propiedades globales de la estructura a la que representa. Por ejemplo, en el caso de las listas, un cierre léxico que implemente la operación *nth* es un objeto localmente efectivo. Sin embargo, si junto al cierre léxico anterior disponemos de otro que implemente la operación *length*, ambos forman un objeto efectivo. En el caso de una estructura de naturaleza infinita, la única posibilidad de codificación es por medio de un objeto localmente efectivo, objeto que aporta menos información que uno efectivo pero que, al menos, permite trabajar con estructuras infinitas y conocer información local de éstas.

Por tanto, la diferencia entre que un objeto sea efectivo o localmente efectivo es del mismo tipo que la diferencia entre la teoría de TADs (algebraica) y la orientación a objetos (coalgebraica). En la literatura clásica, un TAD lleva consigo la característica de generabilidad y se define por medio de signaturas con constructores. El marco de orientación a objetos es puramente observacional: para definir los objetos de una clase se parte de una signatura puramente observacional.

Todas las ideas intuitivas anteriores, que aparecen previamente tanto en el contexto del Cálculo Simbólico en Topología Algebraica ([103], [101], [98], [96]) como en la Teoría de los Lenguajes de Programación ([46], [91], [40], [72], [110], entre otros), se expresan en nuestro contexto tal y como aparecen en la siguiente sección.

4.4 Objetos efectivos

Si pensamos en un objeto efectivo el hecho de que nos permita acceder a información global está relacionado con la posibilidad de construir sus elementos. Así, una posibilidad para definir objetos efectivos pasa por disponer explícitamente de un conjunto de datos, a partir de los que poder construir todos los demás y de los que se puede extraer información de naturaleza global. Una forma de tenerlos explícitamente es por medio de una enumeración almacenada, por ejemplo, en forma de lista finita.

En principio, la característica de efectividad o no puede aplicarse a distintos tipos de objetos. Partiendo de un TAD y llegando a una implementación de éste, resulta que un género de la signatura, en el paso al marco de implementación, se transforma en una representación del conjunto soporte. En definitiva, el problema se traslada en cierta medida a estudiar el carácter efectivo o no de las representaciones de conjuntos, de lo que nos ocupamos en la siguiente sección.

4.4.1 Representaciones efectivas

La noción de representación de un conjunto que hemos introducido y utilizado es la que define una representación como una estructura $\mathcal{R} = (\mathcal{D}_\mathcal{R}, \mathcal{S}_\mathcal{R}, =_\mathcal{R}, \alpha_\mathcal{R}, \mathcal{M}_\mathcal{R})$ con $\mathcal{D}_\mathcal{R}$ un TCCL, $\mathcal{S}_\mathcal{R}$ un subconjunto de éste, $=_\mathcal{R}$ una relación de equivalencia definida sobre los elementos de $\mathcal{S}_\mathcal{R}$, $\alpha_\mathcal{R}$ función de abstracción y $\mathcal{M}_\mathcal{R}$ el conjunto a representar. Si pensamos en representaciones con invariantes e igualdades de representación calculables, la parte de éstas que queda recogida en la máquina está formada por el invariante y la igualdad de la representación, ($invar_\mathcal{R} \colon \mathcal{D}_\mathcal{R} \to \mathcal{U}$, $igual_\mathcal{R} \colon \mathcal{D}_\mathcal{R} \times \mathcal{D}_\mathcal{R} \to \mathcal{U}$) que forman una implementación del PER $(\mathcal{S}_\mathcal{R}, =_\mathcal{R})$ en el dominio $\mathcal{D}_\mathcal{R}$ (según la terminología de Reynolds en [93]). La función $invar_\mathcal{R}$ es total y determina un subconjunto del dominio (concretamente el soporte $\mathcal{S}_\mathcal{R}$), de forma que sobre los elementos de éste está definida la relación de equivalencia dada por la función $igual_\mathcal{R}$. Así, de la parte de representación que queda recogida en la máquina la única información que podemos extraer es local: dado un dato del dominio, saber si está o no en el soporte y, dados dos datos del soporte, saber si son o no iguales en la representación. En este sentido, podemos hablar de *representación localmente efectiva* refiriéndonos a la noción de representación con la que hemos trabajado hasta ahora.

Sin embargo, en el caso de que sean finitos el soporte de la representación o el cociente de éste por la relación de equivalencia definida por la igualdad de la representación, o bien alguno

de ellos pueda generarse a partir de un conjunto finito de datos (en el caso de ser el conjunto subyacente a alguna estructura), desde el punto de vista matemático se tiene un conjunto finito y hay información de carácter global que es conocida, por ejemplo el cardinal (de soporte, del cociente de éste o del conjunto de generadores). Ahora bien, con lo que se tiene hasta ahora esta información no ha quedado recogida en la noción de implementación, no está disponible en la máquina, o lo que es lo mismo, es información que no se conoce de forma constructiva, a partir de los datos de partida. Una forma de solventar lo anterior es introducir esta información en la representación, añadiendo, por ejemplo, una enumeración del soporte, de su cociente por la igualdad de representación o de un conjunto de generadores.

Así, la primera cuestión que nos planteamos formalmente es qué significa dar una enumeración de un conjunto. Si $(A, =_A)$ es un conjunto con igualdad, y $(B, =_B)$ un PER definido sobre él, dar una enumeración de $(B, =_B)$ consiste en dar una constante $card_B\colon \to \mathbb{N}$ que determine el cardinal de $B/_{=_B}$ y una función $enum_B\colon \mathbb{N} \to A$ con dominio $\{n \in \mathbb{N} \mid n < card_B()\}$, de forma que deben verificarse determinadas propiedades respecto a la igualdad $=_B$. Las propiedades que deben verificarse respecto a la igualdad variarán dependiendo de si enumeramos el conjunto B (caso en el que $=_B$ es la heredada como subconjunto de $(A, =_A)$), si lo que enumeramos es un conjunto de representantes canónicos de $B/_{=_B}$ (en el caso de que $=_B$ no coincida con la heredada como subconjunto de $(A, =_A)$) o si es un conjunto de generadores de alguno de ellos.

Volviendo a las representaciones, nuestro punto de partida es el dominio $\mathcal{D}_\mathcal{R}$ y el PER $(\mathcal{S}_\mathcal{R}, =_\mathcal{R})$ sobre $\mathcal{D}_\mathcal{R}$, PER implementado por las funciones $(invar_\mathcal{R}\colon \mathcal{D}_\mathcal{R} \to \mathcal{U}$, $igual_\mathcal{R}\colon \mathcal{D}_\mathcal{R} \times \mathcal{D}_\mathcal{R} \to \mathcal{U})$. Lo primero que nos planteamos es qué significa dar una enumeración en este caso, enumeración que puede serlo de todo el soporte $\mathcal{S}_\mathcal{R}$, de los representantes del cociente $\mathcal{S}_\mathcal{R}/_{=_\mathcal{R}}$ o de un conjunto finito de generadores de alguno de los dos casos anteriores.

Situándonos en el marco Common Lisp, una enumeración es un par de funciones $card_\mathcal{R}\colon \to$ (`satisfies naturalp`) y $enum_\mathcal{R}\colon$ (`satisfies naturalp`) $\to \mathcal{D}_\mathcal{R}$ con dominio {`n` $\in$ `(satisfies naturalp)` | `(< n` $card_\mathcal{R}()$ `)` } verificando determinadas propiedades que dependen de cuál sea el conjunto que se está enumerando.

Así, si pensamos en una enumeración del cociente $\mathcal{S}_\mathcal{R}/_{=_\mathcal{R}}$ deberán verificarse las siguientes propiedades:

i) para todo `n` $\in Def(enum_\mathcal{R})$, se tiene que $invar_\mathcal{R}(enum_\mathcal{R}($`n`$)) \neq_{\texttt{eq}}$ `nil`; es decir $Im(enum_\mathcal{R}) \subseteq \mathcal{S}_\mathcal{R}$

ii) para todo `n1`, `n2` $\in Def(enum_\mathcal{R})$ con `n1` $\neq$ `n2`, se tiene que $igual_\mathcal{R}(enum_\mathcal{R}($`n1`$), enum_\mathcal{R}($`n2`$)) =_{\texttt{eq}}$ `nil`

iii) para todo `d` $\in \mathcal{D}_\mathcal{R}$ tal que $invar_\mathcal{R}($`d`$) \neq_{\texttt{eq}}$ `nil`, existe `n` $\in Def(enum_\mathcal{R})$ tal que $igual_\mathcal{R}($`d`$, enum_\mathcal{R}($`n`$)) \neq_{\texttt{eq}}$ `nil`

La condición exigida en *i*) garantiza que los datos que enumeramos son datos del soporte de la representación. La condición *ii*) garantiza que solo se enumeran, como mucho, un dato de cada clase de equivalencia de $\mathcal{S}_\mathcal{R}/_{=_\mathcal{R}}$. Finalmente, la condición *iii*) garantiza que todas las clases de equivalencia de $\mathcal{S}_\mathcal{R}/_{=_\mathcal{R}}$ poseen, al menos, un representante en la enumeración. De las dos últimas se desprende la unicidad del dato `n` que aprarece en *iii*), es decir, de *ii*) y *iii*) se desprende que la enumeración está formada por exactamente un representante de cada clase de equivalencia de $\mathcal{S}_\mathcal{R}/_{=_\mathcal{R}}$.

Sin embargo, si pensamos en enumerar todo el soporte, las condiciones que deben verificarse son otras y pueden expresarse del siguiente modo:

i) para todo $\mathtt{n} \in Def(enum_{\mathcal{R}})$, se tiene que $invar_{\mathcal{R}}(enum_{\mathcal{R}}(\mathtt{n})) \neq_{\mathtt{eq}} \mathtt{nil}$

ii) para todo $\mathtt{n1}, \mathtt{n2} \in Def(enum_{\mathcal{R}})$ con $\mathtt{n1} \neq \mathtt{n2}$, se verifica $enum_{\mathcal{R}}(\mathtt{n1}) \neq_{\mathcal{D}_{\mathcal{R}}} enum_{\mathcal{R}}(\mathtt{n2})$

iii) $card_{\mathcal{R}}()$ coincide con el cardinal del soporte.

Hay dos diferencias con respecto a lo exigido a las enumeraciones de representantes del cociente $\mathcal{S}_{\mathcal{R}}/ =_{\mathcal{R}}$. Por un lado, la inyectividad exigida a la función $enum_{\mathcal{R}}$ (propiedad recogida en *ii*)), en el primer caso es respecto a la igualdad de la representación, mientras que en el segundo caso es respecto a la igualdad heredada como subconjunto del dominio de la representación. Por otro lado, en este segundo caso la condición *iii*) unida a la *ii*) garantiza que la imagen de la función $enum_{\mathcal{R}}$ cubre (es exactamente) el soporte de la representación. Es claro que en este caso, las condiciones *i*) y *iii*) son equivalentes a $Im(enum_{\mathcal{R}}) = \mathcal{S}_{\mathcal{R}}$.

También es posible encontrarnos en la situación de conocer un conjunto de datos a partir de los que generar todo el soporte (o un conjunto de representantes canónicos del cociente de éste por la igualdad de la representación). Este caso es muy general en el sentido de que el término "generador" lo estamos entendiendo en el sentido más amplio, entendiendo que a partir de un conjunto mínimo de datos podemos obtener todos. La forma en la que se obtengan, es decir, la estrategia para generarlos, depende de cada ejemplo concreto (recordar los ejemplos ya vistos de los símplices abstractos en el caso de los conjuntos simpliciales y las combinaciones en el caso de los complejos de cadenas). Nuestro estudio teórico se va a centrar en los dos primeros casos de enumeración, el caso de generabilidad lo recogeremos al final en un ejemplo concreto, de forma que quede patente la posibilidad de aplicar el mismo procedimiento obteniendo los mismos resultados a cualquier estrategia de generabilidad.

Enriquecemos a continuación la noción de representación recogiendo en ésta más información acerca del conjunto que se representa. Concretamente, introducimos un nuevo tipo de representaciones que recogen enumeraciones explícitas.

Definición 4.4.1 *Una* representación efectiva *de un conjunto $\mathcal{M}$ es una estructura $\mathcal{R}^{ef} = (\mathcal{R}, card_{\mathcal{R}}, enum_{\mathcal{R}})$ siendo $\mathcal{R} = (\mathcal{D}_{\mathcal{R}}, \mathcal{S}_{\mathcal{R}}, =_{\mathcal{R}}, \alpha_{\mathcal{R}}, \mathcal{M})$ una representación con invariante e igualdad de representación calculables y función de abstracción sobreyectiva y $(card_{\mathcal{R}}, enum_{\mathcal{R}})$ una enumeración de $\mathcal{S}_{\mathcal{R}}$ o de $\mathcal{S}_{\mathcal{R}}/_{=_{\mathcal{R}}}$.*

El exigir que la función de abstracción sea sobreyectiva es para que todos los elementos del modelo estén representados, algo natural al trabajar con conjuntos de naturaleza finita.

Una forma de obtener representaciones efectivas de un conjunto $\mathcal{M}$ de naturaleza finita es completando una representación de $\mathcal{M}$ con función de abstracción sobreyectiva, con una enumeración del soporte o del cociente de éste por la igualdad de la representación.

Ejemplo 4.4.2 Consideramos el conjunto $\mathbb{Z}/n\mathbb{Z}$ para algún $n \in \mathbb{N}$ con $n > 1$. Vamos a dar varias representaciones efectivas para el conjunto anterior.

1. Consideramos la representación $\mathcal{R}_1$ que tiene por dominio el tipo `integer`, como soporte también `integer`, igualdad de representación la igualdad módulo n (es decir, `i1` $=_{\mathcal{R}_1}$ `i2` si y solo si `(= (mod i1 n) (mod i2 n))`) y como función de abstracción la que a cada dato `i` de `integer` lo lleva a la clase $[i \ mod \ n]$. La representación anterior posee invariante e igualdad de representación calculables y la función de abstracción es sobreyectiva. $\mathcal{R}_1$ es una representación localmente efectiva del conjunto $\mathbb{Z}/n\mathbb{Z}$.

   Consideramos las siguientes funciones

- $card_{\mathcal{R}_1}$: $\rightarrow$ (`satisfies naturalp`) con imagen la constante `n`
- $enum_{\mathcal{R}_1}$: (`satisfies naturalp`) $\rightarrow$ `integer` con dominio el conjunto $\{$`i` $\in$ `(satisfies naturalp)` | `(<= 0 i)` e `(< i n)`$\}$ y comportamiento $enum_{\mathcal{R}_1}$(`i`) := `i`

$\mathcal{R}_1^{ef} = (\mathcal{R}_1, card_{\mathcal{R}_1}, enum_{\mathcal{R}_1})$ es una representación efectiva del conjunto $\mathbb{Z}/n\mathbb{Z}$ con enumeración de representantes del cociente del soporte por la igualdad de la representación.

2. Definimos $\mathcal{R}_2$ como la representación con dominio `integer`, soporte el conjunto `(mod n)` ($\{0, \ldots n-1\}$), igualdad de representación la del tipo, es decir, la dada por la igualdad Lisp `=`, y función de abstracción la que a cada `i` del soporte lo lleva a la clase $[i]$. $\mathcal{R}_2^{ef} = (\mathcal{R}_2, card_{\mathcal{R}_1}, enum_{\mathcal{R}_1})$ es una representación efectiva del conjunto $\mathbb{Z}/n\mathbb{Z}$.

3. Podemos dar una tercera representación que denotamos por $\mathcal{R}_3$ del conjunto $\mathbb{Z}/n\mathbb{Z}$, representación con dominio `integer`, soporte el conjunto $\{0, \ldots, 2n-1\}$, igualdad de representación la igualdad módulo `n` y función de abstracción la que a cada `i` del soporte lo lleva a la clase $[i \ mod \ n]$. $\mathcal{R}_3^{ef} = (\mathcal{R}_3, card_{\mathcal{R}_1}, enum_{\mathcal{R}_1})$ es también representación efectiva del conjunto $\mathbb{Z}/n\mathbb{Z}$.

   Si consideramos la función $enum_{\mathcal{R}_3}$: (`satisfies naturalp`) $\rightarrow$ `integer` con dominio $\{$`i` $\in$ `integer` | `(<= 0 i)` e `(< i n)`$\}$ y comportamiento $enum_{\mathcal{R}_1}$(`i`) := `i` + `n`, la terna $\mathcal{R}_3^{ef} = (\mathcal{R}_3, card_{\mathcal{R}_1}, enum_{\mathcal{R}_3})$ es también representación efectiva del conjunto $\mathbb{Z}/n\mathbb{Z}$.

4. Consideramos la representación $\mathcal{R}_3$ anterior y vamos a completarla hasta una representación efectiva con enumeración de todos los datos del soporte. Para ello definimos las siguientes funciones:

   - $card_{\mathcal{R}_3'}$: $\rightarrow$ (`satisfies naturalp`) con imagen la constante `2n`
   - $enum_{\mathcal{R}_3'}$: (`satisfies naturalp`) $\rightarrow$ `integer` con dominio $\{$`i` $\in$ `integer` | `(<= 0 i)` e `(< i 2n)`$\}$ y comportamiento $enum_{\mathcal{R}_1}$(`i`) := `i`

   La terna $(\mathcal{R}_3, card_{\mathcal{R}_3'}, enum_{\mathcal{R}_3'})$ es una representación efectiva del conjunto $\mathbb{Z}/n\mathbb{Z}$ con enumeración de todos los elementos del soporte.

□

A partir de una representación efectiva podemos obtener información global del conjunto al que representa. En particular, a partir de la enumeración, podemos conocer si es o no vacío.

Introducida la noción de representación efectiva, el siguiente paso natural es definir morfismos entre ellas.

Definición 4.4.3 *Si* $\mathcal{R}_1^{ef} = (\mathcal{R}_1, card_{\mathcal{R}_1}, enum_{\mathcal{R}_1})$ *y* $\mathcal{R}_2^{ef} = (\mathcal{R}_2, card_{\mathcal{R}_2}, enum_{\mathcal{R}_2})$ *son representaciones efectivas, una* r–transformación (t, g): $\mathcal{R}_1^{ef} \rightarrow \mathcal{R}_2^{ef}$ *entre ellas es una r–tranformación* (t, g): $\mathcal{R}_1 \rightarrow \mathcal{R}_2$ *que conmuta con las enumeraciones, es decir, verifica:*

i) $card_{\mathcal{R}_1}() = card_{\mathcal{R}_2}()$

ii) $t \circ enum_{\mathcal{R}_1} = enum_{\mathcal{R}_2}$

La conmutatividad de una r–transformación con los invariantes y las igualdades de representación se tiene por la propia definición de r–transformación, por ello en la definición anterior solo es necesario exigir que conmute con las enumeraciones.

Las representaciones efectivas junto con las r–transformaciones entre ellas forman una categoría.

4.4.2 Especificando implementaciones sobre representaciones efectivas

Vamos a estudiar a continuación las implementaciones sobre representaciones efectivas. Guiados por lo ya desarrollado en el capítulo anterior y en la primera parte de éste, el objetivo final es dar una implementación de una determinada álgebra en la que queden recogidas todas las implementaciones sobre representaciones efectivas de una categoría concreta.

Sea Σ una signatura y Σ_{imp} la signatura obtenida aplicando a Σ la operación $()_{imp}$. Apoyándonos en la forma de trabajar en la primera parte de este capítulo, se trata de definir una signatura que recoja parte de las representaciones. En el caso anterior, definimos la operación $()_{rep}$ que añadía a cada signatura Σ_{imp} dos operaciones por cada género de Σ, una que implementa al invariante y otra a la igualdad de representación del conjunto soporte del género. En este caso la idea es la misma, añadir dos operaciones por cada enumeración de un conjunto soporte, una que refleja el cardinal de la enumeración y la otra es la enumeración explícita. Sin embargo, en la mayor parte de los ejemplos no interesa que las representaciones de todos los géneros de una signatura sean efectivas. La cuestión es que, las representaciones efectivas solo pueden serlo de conjuntos de naturaleza finita y, en la mayor parte de los ejemplos, esto no ocurre para todos los géneros. Por ejemplo, en el caso de los conjuntos simpliciales, podemos pensar en partir de un número finito de símplices geométricos en cada dimensión y considerar una representación efectiva, pero para el género *nat* que actúa como índice, el conjunto soporte es $\mathbb{N}$ que no es de naturaleza finita (teniendo en cuenta que no lo estamos generando, sino que lo tomamos simplemente como conjunto). Lo mismo ocurre en el caso de los complejos de cadenas. Si trabajamos con complejos de cadenas finitamente generados en cada grado, es posible recoger una enumeración de los generadores para el género correspondiente. Sin embargo, el conjunto soporte para el género *int* que representa a los enteros que aparecen como coeficientes en las combinaciones lineales tampoco es finitamente generado (en la signatura en la que estamos). Por ello, no resulta conveniente, a partir de una signatura, exigir enumeración para todos los géneros.

Esto nos lleva a considerar, para cada signatura Σ, un subconjunto $G^{\diamond}$ de G de forma que las Σ-álgebras que se consideren tendrán conjunto soporte de naturaleza finita para los géneros de $G^{\diamond}$.

Definimos la operación $()_{ef}$ que construye a partir de una signatura Σ_{imp} la signatura que denotaremos por $\Sigma_{imp_{ef}}$ definida como sigue:

- $G_{imp_{ef}} = G_{imp_{rep}} \cup \{nat\}$
- $\Omega_{imp_{ef}} = \Omega_{imp_{rep}} \cup \{rep\_card_g \colon imp_\Sigma \to nat\}_{g \in G^{\diamond}} \cup \{rep\_enum_g \colon imp_\Sigma\ nat \to g\}_{g \in G^{\diamond}}$

El género *nat* se utilizará para numerar datos. Si en Σ ya existe un género con esa intención, se renombra a *nat*. Observar que las operaciones de esta signatura están definidas a partir de las de $\Omega_{imp_{rep}}$ donde ya queda incluida la parte de representación correspondiente a invariantes e igualdades. Por eso, en la definición anterior explícitamente solo aparece lo correspondiente a las enumeraciones.

Abusando de la notación, la interpretación de cada operación rep_card_g en un álgebra A, $(rep\_card_g)_A$, la denotaremos por $rep\_card_{A_g}$ y, análogamente para las operaciones rep_enum_g.

La signatura $\Sigma_{imp_{ef}}$ recoge la parte sintáctica de la especificación de familias de álgebras que provienen de implementaciones sobre representaciones efectivas. Por ello, en este momento, a partir de un TAD $\mathcal{T}$ nuestro interés no radica en estudiar toda su categoría de implementaciones, sino que solo nos interesan aquellas definidas sobre representaciones efectivas y para las que los dominios de los programas que implementan a las operaciones de las Σ-álgebras pueden definirse a partir de los invariantes, igualdades y enumeraciones de las representaciones. Denotamos por $Imp_{ef}(\mathcal{T})$ a una subcategoría no plena de la categoría de implementaciones del TAD $\mathcal{T}$ con objetos las implementaciones sobre representaciones efectivas para los géneros de $G^{\diamond}$, con programas que implementan a las operaciones totales sobre el producto de los soportes de las representaciones correspondientes y con invariantes, igualdades y enumeraciones de las representaciones explícitos. Como morfismos de la categoría $Imp_{ef}(\mathcal{T})$ tomamos las i–transformaciones que para los géneros de $G^{\diamond}$ como r–transformaciones lo son entre representaciones efectivas, es decir, son r–transformaciones que conmutan con las enumeraciones.

Así, cada implementación $\mathcal{J} = (\underline{\mathcal{R}}, (\mathtt{p}^{\mathcal{J}}_{\sigma})_{\sigma\in\Omega})$ de la categoría $Imp_{ef}(\mathcal{T})$ tiene asociadas cuatro familias de programas, una de ellas formada por los programas que implementan a los invariantes, otra por los que implementan a las igualdades de las representaciones, y las dos últimas implementan a las enumeraciones de los soportes de las representaciones o sus cocientes por las igualdades de las representaciones. Así, una tal implementación podemos verla como una estructura: $\mathcal{J} = \big(\underline{\mathcal{R}}, (\mathtt{p}^{\mathcal{J}}_{invar_{\mathcal{R}_g}})_{g\in G}, (\mathtt{p}^{\mathcal{J}}_{igual_{\mathcal{R}_g}})_{g\in G}, (\mathtt{p}^{\mathcal{J}}_{card_{\mathcal{R}_g}})_{g\in G^{\diamond}}, (\mathtt{p}^{\mathcal{J}}_{enum_{\mathcal{R}_g}})_{g\in G^{\diamond}}, (\mathtt{p}^{\mathcal{J}}_{\sigma})_{\sigma\in\Omega}\big)$, siendo $\mathcal{R}_g$ representación efectiva para cada $g \in G^{\diamond}$, y localmente efectiva para $g \in G \setminus G^{\diamond}$; y, siendo para cada $g \in G$, $\mathtt{p}^{\mathcal{J}}_{invar_{\mathcal{R}_g}}$ un programa que implementa al invariante de la representación $\mathcal{R}_g$ y $\mathtt{p}^{\mathcal{J}}_{igual_{\mathcal{R}_g}}$ uno que implementa a la igualdad de la representación $\mathcal{R}_g$; y, para cada $g \in G^{\diamond}$, $\mathtt{p}^{\mathcal{J}}_{card_{\mathcal{R}_g}}$ y $\mathtt{p}^{\mathcal{J}}_{enum_{\mathcal{R}_g}}$ son programas que implementan, respectivamente, el cardinal del soporte de $\mathcal{R}_g$ o su cociente por la igualdad de la representación y una enumeración de los elementos del mismo. El hecho de que los programas anteriores implementen funciones como el invariante, igualdad de representación y enumeraciones, nos permite deducir algunas propiedades de éstos:

- Para cada $g \in G$, el programa $\mathtt{p}^{\mathcal{J}}_{invar_{\mathcal{R}_g}}$ que implementa al invariante tiene como tipo de entrada $\mathcal{D}_g$, dominio de la representación $\mathcal{R}_g$, $\mathcal{U}$ como tipo de salida, $\mathcal{D}_g$ como dominio de definición y código $\mathtt{c}^{\mathtt{p}^{\mathcal{J}}_{invar_{\mathcal{R}_g}}}$ que verifica, para cada dato $\mathtt{d}$ del dominio $\mathcal{D}_g$, que $\mathtt{c}^{\mathtt{p}^{\mathcal{J}}_{invar_{\mathcal{R}_g}}}(\mathtt{d}) \neq_{\mathtt{eq}} \mathtt{nil}$ si y solo si $\mathtt{d}$ pertenece al soporte $\mathcal{S}_g$.
- Para cada $g \in G$, $igual_{\mathcal{R}_g} \colon \mathcal{D}_g \times \mathcal{D}_g \to \mathcal{U}$ es una función parcial con dominio $\mathcal{S}_g \times \mathcal{S}_g$ que define la igualdad de la representación. Así, el código $\mathtt{c}^{\mathtt{p}^{\mathcal{J}}_{igual_{\mathcal{R}_g}}}$ del programa $\mathtt{p}^{\mathcal{J}}_{igual_{\mathcal{R}_g}}$ verifica, para cada par $\mathtt{d}, \mathtt{d}' \in \mathcal{S}_g$, que $\mathtt{c}^{\mathtt{p}^{\mathcal{J}}_{igual_{\mathcal{R}_g}}}(\mathtt{d}, \mathtt{d}') \neq_{\mathtt{eq}} \mathtt{nil}$ si y solo si $\mathtt{d} =_{\mathcal{R}_g} \mathtt{d}'$.
- La función $card_{\mathcal{R}_g} \colon \to$ (`satisfies naturalp`) determina el cardinal de la enumeración de datos. El programa $\mathtt{p}^{\mathcal{J}}_{card_{\mathcal{R}_g}}$ que la implementa tiene al tipo vacío `nil` como tipo de entrada, a `(satisfies naturalp)` de salida, `nil` como soporte y es tal que $F(\mathtt{p}^{\mathcal{J}}_{card_{\mathcal{R}_g}})()$ coincide con el cardinal de $\mathcal{S}_g$ o con el de $\mathcal{S}_g/=_{\mathcal{R}_g}$.
- La función $enum_{\mathcal{R}_g} \colon$ (`satisfies naturalp`) $\to \mathcal{D}_{\mathtt{g}}$ parcial con dominio $\big\{\mathtt{n} \in$ `(satisfies naturalp)` $\mid$ `(<` `n` $card_{\mathcal{R}_g}()$`)`$\big\}$, es implementada por el programa $\mathtt{p}^{\mathcal{J}}_{enum_{\mathcal{R}_g}}$ que tiene a `(satisfies naturalp)` como tipo de entrada y a $\mathcal{D}_g$ como tipo de salida; el soporte $\mathtt{S}^{\mathtt{p}^{\mathcal{J}}_{enum_{\mathcal{R}_g}}}$ es el conjunto $\big\{\mathtt{n} \in$ `(satisfies naturalp)` $\mid$ $\mathtt{n} < F(\mathtt{p}^{\mathcal{J}}_{card_{\mathcal{R}_g}})()\big\}$; y, el código

$\mathtt{c}^{\mathtt{p}^{\mathcal{J}}_{enum_{\mathcal{R}_g}}}$ debe verificar las condiciones correspondientes a las enumeraciones, así, dependiendo de qué se esté enumerando, si $\mathcal{S}_g/=_{\mathcal{R}_g}$ o $\mathcal{S}_g$, se deben verificar uno de los dos bloques siguientes de condiciones. El siguiente en el caso de enumerar $\mathcal{S}_g/=_{\mathcal{R}_g}$

i) para todo $\mathtt{n} \in \mathtt{S}^{\mathtt{p}^{\mathcal{J}}_{enum_{\mathcal{R}_g}}}$ se tiene que $F(\mathtt{p}^{\mathcal{J}}_{invar_{\mathcal{R}_g}})\big(F(\mathtt{p}^{\mathcal{J}}_{enum_{\mathcal{R}_g}})(\mathtt{n})\big) \neq_{\mathtt{eq}} \mathtt{nil}$

ii) para todo $\mathtt{n1}, \mathtt{n2} \in \mathtt{S}^{\mathtt{p}^{\mathcal{J}}_{enum_{\mathcal{R}_g}}}$ con $\mathtt{n1} \neq \mathtt{n2}$ se tiene que

$$F(\mathtt{p}^{\mathcal{J}}_{igual_{\mathcal{R}_g}})\big(F(\mathtt{p}^{\mathcal{J}}_{enum_{\mathcal{R}_g}})(\mathtt{n1}), F(\mathtt{p}^{\mathcal{J}}_{enum_{\mathcal{R}_g}})(\mathtt{n2})\big) =_{\mathtt{eq}} \mathtt{nil}$$

iii) para todo $\mathtt{d} \in \mathcal{D}_g$ tal que $F(\mathtt{p}^{\mathcal{J}}_{invar_{\mathcal{R}_g}})(\mathtt{d}) \neq_{\mathtt{eq}} \mathtt{nil}$, existe $\mathtt{n} \in \mathtt{S}^{\mathtt{p}^{\mathcal{J}}_{enum_{\mathcal{R}_g}}}$ tal que $F(\mathtt{p}^{\mathcal{J}}_{igual_{\mathcal{R}_g}})\big(\mathtt{d}, F(\mathtt{p}^{\mathcal{J}}_{enum_{\mathcal{R}_g}})(\mathtt{n})\big) \neq_{\mathtt{eq}} \mathtt{nil}$.

y en el caso de enumerar $\mathcal{S}_g$, el bloque de condiciones es

i') para todo $\mathtt{n} \in \mathtt{S}^{\mathtt{p}^{\mathcal{J}}_{enum_{\mathcal{R}_g}}}$ se tiene que $F(\mathtt{p}^{\mathcal{J}}_{invar_{\mathcal{R}_g}})\big(F(\mathtt{p}^{\mathcal{J}}_{enum_{\mathcal{R}_g}})(\mathtt{n})\big) \neq_{\mathtt{eq}} \mathtt{nil}$

ii') para todo $\mathtt{n1}, \mathtt{n2} \in \mathtt{S}^{\mathtt{p}^{\mathcal{J}}_{enum_{\mathcal{R}_g}}}$ con $\mathtt{n1} \neq \mathtt{n2}$ se verifica $F(\mathtt{p}^{\mathcal{J}}_{enum_{\mathcal{R}_g}})(\mathtt{n1}) \neq_{\mathcal{D}_g} F(\mathtt{p}^{\mathcal{J}}_{enum_{\mathcal{R}_g}})(\mathtt{n2})$

iii') el cardinal del soporte $\mathcal{S}_g$ es justamente $F(\mathtt{p}^{\mathcal{J}}_{card_{\mathcal{R}_g}})()$.

- Además, para cada $(\sigma\colon \omega \to v) \in \Sigma$ con $\omega = g_1 \dots g_n$, la función definida por el programa $\mathtt{p}^{\mathcal{J}}_{\sigma}$ tiene por dominio $\mathcal{S}^{\mathcal{J}}_{g_1} \times \cdots \times \mathcal{S}^{\mathcal{J}}_{g_n}$

Vamos a considerar $\Sigma_{imp_{ef}}$-álgebras que recogen información sobre familias de implementaciones de la categoría $Imp_{ef}(\mathcal{T})$.

Definimos la categoría $CImp_{ef}(\mathcal{T}_{imp})$ como sigue:

- Los objetos son las $\Sigma_{imp_{ef}}$-álgebras $A = \big\langle \mathtt{T}_{imp_\Sigma}, (\mathtt{T}_g)_{g\in G}, \mathtt{T}_{bool}, \mathtt{T}_{nat}, \{imp\_\sigma_A\colon \mathtt{T}_{imp_\Sigma} \times \mathtt{T}_\omega \to \mathtt{T}_v, Def(imp\_\sigma_A)\}_{(\sigma\colon \omega\to v)\in\Omega}, \{rep\_invar_{A_g}\colon \mathtt{T}_{imp_\Sigma} \times \mathtt{T}_g \to \mathtt{T}_{bool}, Def(rep\_invar_{A_g})\}_{g\in G}, \{rep\_igual_{A_g}\colon \mathtt{T}_{imp_\Sigma} \times \mathtt{T}_g \times \mathtt{T}_g \to \mathtt{T}_{bool}, Def(rep\_igual_{A_g})\}_{g\in G}, \{rep\_card_{A_g}\colon \mathtt{T}_{imp_\Sigma} \to \mathtt{T}_{nat}, Def(rep\_card_{A_g})\}_{g\in G^\diamond}, \{rep\_enum_{A_g}\colon \mathtt{T}_{imp_\Sigma} \times \mathtt{T}_{nat} \to \mathtt{T}_g, Def(rep\_enum_{A_g})\}_{g\in G^\diamond}\big\rangle$ tales que, para cada Σ-álgebra $A_{\mathtt{d}} = \big\langle (\mathtt{T}_g)_{g\in G}, \{imp\_\sigma_A(\mathtt{d}, -)\colon \mathtt{T}_\omega \to \mathtt{T}_v, Def(imp\_\sigma_A(\mathtt{d}, -))\}_{(\sigma\colon \omega\to v)\in\Sigma}\big\rangle$ existe $\mathcal{J}$ implementación de la categoría $Imp_{ef}(\mathcal{T})$ con G-tipo $\underline{\mathtt{T}}$, invariantes e igualdades de representación dados por $rep\_invar_{A_g}(\mathtt{d}, -)$ y $rep\_igual_{A_g}(\mathtt{d}, -)$, respectivamente, para cada $g \in G$, enumeraciones de las representaciones dadas, para cada $g \in G^\diamond$ por $rep\_card_{A_g}(\mathtt{d}, -)$ y $rep\_enum_{A_g}(\mathtt{d}, -)$, y, funciones definidas por los programas que implementan a las operaciones de la Σ-álgebra totales sobre los productos de los soportes de las representaciones y definidas por las funciones $imp\_\sigma_A(\mathtt{d}, -)$, para cada $\sigma \in \Sigma$.

  De las condiciones anteriores se desprende que, $F(\mathtt{p}^{\mathcal{J}}_{invar_{\mathcal{R}_g}}) = rep\_invar_{A_g}(\mathtt{d}, -)$ y $F(\mathtt{p}^{\mathcal{J}}_{igual_{\mathcal{R}_g}}) = rep\_igual_{A_g}(\mathtt{d}, -)$, para cada $g \in G$; $F(\mathtt{p}^{\mathcal{J}}_{card_{\mathcal{R}_g}}) = rep\_card_{A_g}(\mathtt{d}, -)$ y $F(\mathtt{p}^{\mathcal{J}}_{enum_{\mathcal{R}_g}}) = rep\_enum_{A_g}(\mathtt{d}, -)$, para cada $g \in G^\diamond$; y, para cada $\sigma \in \Sigma$, $F(\mathtt{p}^{\mathcal{J}}_{\sigma}) = imp\_\sigma_A(\mathtt{d}, -)$.

- Si A y B son $\Sigma_{imp_{ef}}$-álgebras de la categoría $\mathcal{C}Imp_{ef}(\mathcal{T}_{imp})$, un $\Sigma_{imp_{ef}}$-morfismo $f\colon A \to B$, $f = (f_{imp_\Sigma}, (f_g)_{g\in G}, f_{bool}, f_{nat})$, es un morfismo de $\mathcal{C}Imp_{ef}(\mathcal{T}_{imp})$ si, para todo $\mathtt{d} \in A_{imp_\Sigma}$, $(f_g)_{g\in G}\colon A_{\mathtt{d}} \to B_{f_{imp_\Sigma}(\mathtt{d})}$ es un morfismo entre las Σ-álgebras y, además se verifican las siguientes condiciones:

  · para cada $g \in G$, $f_g(\mathcal{S}_g^{\mathcal{J}^A}) \subseteq \mathcal{S}_g^{\mathcal{J}^B}$;

  · para cada $g \in G$, $f_{bool} \circ rep\_invar_{A_g}(\mathtt{d}, -) = rep\_invar_{B_g}(f_{imp_\Sigma}(\mathtt{d}), -) \circ f_g$;

  · para cada $g \in G$, $f_{bool} \circ rep\_igual_{A_g}(\mathtt{d}, -) = rep\_igual_{B_g}(f_{imp_\Sigma}(\mathtt{d}), -) \circ (f_g, f_g)$.

  · para cada $g \in G^\diamond$, $f_{nat} \circ rep\_card_{A_g}(\mathtt{d}, -) = rep\_card_{B_g}$;

  · para cada $g \in G^\diamond$, $f_g \circ rep\_enum_{A_g}(\mathtt{d}, -) = rep\_enum_{B_g}(f_{imp_\Sigma}(\mathtt{d}), -) \circ f_{nat}$.

  Las condiciones anteriores se exigen para que haya conmutatividad no solo sobre los operadores de las Σ-álgebras sino también sobre los que definen los invariantes, igualdades y enumeraciones de las representaciones.

Es claro que cada $\Sigma_{imp_{ef}}$-álgebra de $\mathcal{C}Imp_{ef}(\mathcal{T}_{imp})$ recoge una familia de implementaciones sobre representaciones efectivas del TAD $\mathcal{T}$ con el mismo G-tipo y con funciones definidas por los programas totales sobre los soportes de las representaciones. Por la definición que hemos dado de enumeración, todas las $\Sigma_{imp_{ef}}$-álgebras de $\mathcal{C}Imp_{ef}(\mathcal{T}_{imp})$ tienen por conjunto soporte para el género *nat* el tipo `(satisfies naturalp)`. Sin pérdida de generalidad, fijamos $\mathcal{U}$ como soporte para el género *bool* con la igualdad que interpreta cualquier objeto distinto de `nil` (por `eq`) como el valor booleano *true*.

Ejemplo 4.4.4 Retomamos el ejemplo 4.2.1 en el que definimos una $\mathtt{GRP}_{imp_{rep}}$-álgebra que recoge una implementación de cada grupo cociente de $\mathbb{Z}$ módulo n, con $n > 1$, implementación que denotamos por $\mathcal{I}^n_{\mathtt{GRP}}$ y para la que definimos programas que implementaban a los invariantes y a las igualdades de las representaciones. Vamos a enriquecer cada una de las representaciones sobre las que se definen esas implementaciones obteniendo así representaciones efectivas. Para cada $n > 1$, la representación $\mathcal{R}^n_{\mathtt{GRP}}$ de $\mathbb{Z}/n\mathbb{Z}$ sobre la que se define $\mathcal{I}^n_{\mathtt{GRP}}$ tiene por dominio `integer`, `(mod n)` como soporte y función de abstracción la que a cada `i` del soporte lo lleva a la clase del propio i. Al no dar la igualdad de representación explícitamente, se está considerando la propia del tipo. Definimos las siguientes funciones:

· la función $card_{\mathcal{R}^n_{\mathtt{GRP}}}\colon \to$ (`satisfies naturalp`) definida como la constante `n`; y,

· la función $enum_{\mathcal{R}^n_{\mathtt{GRP}}}$: (`satisfies naturalp`) $\to$ `integer` es parcial con dominio el conjunto $\{\mathtt{i} \in (\mathtt{satisfies\ naturalp}) \mid (\mathtt{<\ i\ n})\}$, y comportamiento $enum_{\mathcal{R}^n_{\mathtt{GRP}}}(\mathtt{i}) := \mathtt{i}$.

La terna $\mathcal{R}^{ef,n}_{\mathtt{GRP}} = (\mathcal{R}^n_{\mathtt{GRP}}, card_{\mathcal{R}^n_{\mathtt{GRP}}}, enum_{\mathcal{R}^n_{\mathtt{GRP}}})$ es una representación efectiva del conjunto $\mathbb{Z}/n\mathbb{Z}$. La representación efectiva $\mathcal{R}^{ef,n}_{\mathtt{GRP}}$ junto con los programas de la implementación $\mathcal{I}^n_{\mathtt{GRP}}$ definen una implementación de $\mathbb{Z}/n\mathbb{Z}$ que denotamos por $\mathcal{I}^{ef,n}_{\mathtt{GRP}}$.

La siguiente signatura corresponde a la signatura $\mathtt{GRP}_{imp_{ef}}$

$$
\begin{array}{lcllllll}
imp\_prd & : & imp_{\texttt{GRP}} & g & g & \rightarrow & g \\
imp\_inv & : & imp_{\texttt{GRP}} & g & & \rightarrow & g \\
imp\_unt & : & imp_{\texttt{GRP}} & & & \rightarrow & g \\
rep\_invar & : & imp_{\texttt{GRP}} & g & & \rightarrow & bool \\
rep\_igual & : & imp_{\texttt{GRP}} & g & g & \rightarrow & bool \\
rep\_card & : & imp_{\texttt{GRP}} & & & \rightarrow & nat \\
rep\_igual & : & imp_{\texttt{GRP}} & nat & & \rightarrow & g
\end{array}
$$

Una $\texttt{GRP}_{imp_{ef}}$-álgebra que recoge a la familia de implementaciones $\{\mathcal{I}^{ef,n}_{\texttt{GRP}}\}_{n\in\mathbb{N},n>1}$ es la $\texttt{GRP}_{imp_{ef}}$-álgebra B, definida completando la $\texttt{GRP}_{imp_{rep}}$-álgebra A (del ejemplo 4.2.1) del siguiente modo:

- Como conjunto soporte para el género *nat* se toma `(satisfies naturalp)`.
- La función rep_card_B: `(satisfies positivoMayorQue1p)` $\rightarrow$ `(satisfies naturalp)` es total y está definida por rep_card_B(`n`):= `n`.
- La función rep_enum_B: `(satisfies positivoMayorQue1p)` $\times$ `(satisfies naturalp)` $\rightarrow$ `integer` es parcial con dominio $\{$(`n`, `i`) | `(< i n)`$\}$ y definida por rep_enum_B(`n`, `i`):= `i`.

La familia de implementaciones $\{\mathcal{I}^{ef,n}_{\texttt{GRP}}\}_{n\in\mathbb{N},n>1}$ garantiza que la $\texttt{GRP}_{imp_{ef}}$-álgebra B es objeto de la categoría $CImp_{ef}(\mathcal{T}_{\texttt{GRP}_{imp}})$. Además, en el ejemplo 4.2.1 vimos programas que implementan, para cada `n` $\in$ `(satisfies positivoMayorQue1p)`, al invariante y a la igualdad de representación de $\mathcal{I}^{ef,n}_{\texttt{GRP}}$. A continuación damos programas que implementan a las enumeraciones:

- $\texttt{p}^{\texttt{n}}_{card} = (\texttt{c}^{\texttt{p}^{\texttt{n}}_{card}}, \texttt{nil}, \texttt{nil}, \texttt{(satisfies naturalp)})$ con código el siguiente objeto funcional

  $\texttt{c}^{\texttt{p}^{\texttt{n}}_{card}} \equiv$ `#'(lambda ( ) n)`

- $\texttt{p}^{\texttt{n}}_{enum}$=($\texttt{c}^{\texttt{p}^{\texttt{n}}_{enum}}$, `(mod n)`, `(satisfies naturalp)`, `integer`) siendo su código el siguiente objeto funcional

  $\texttt{c}^{\texttt{p}^{\texttt{n}}_{enum}} \equiv$ `#'(lambda (i) i)`

□

Siguiendo el mismo esquema que en todo el trabajo, descomponemos la categoría $\mathcal{C}Imp_{ef}(\mathcal{T}_{imp})$ en subcategorías con conjuntos soporte fijos para considerar familias de implementaciones que trabajan sobre los mismos datos. Fijamos un G-tipo $\underline{\texttt{T}} = (\texttt{T}_g)_{g\in G}$ y denotamos

por $\mathcal{C}Imp_{ef}^{\mathtt{T},\{\}}(\mathcal{T}_{imp})$ a la subcategoría de $\mathcal{C}Imp_{ef}(\mathcal{T}_{imp})$ con objetos los que tienen como conjuntos soporte para los géneros de Σ el G-tipo $\underline{\mathtt{T}}$ y morfismos los de $\mathcal{C}Imp_{ef}(\mathcal{T}_{imp})$ que son la identidad sobre todos los géneros excepto imp_Σ.

Por abreviar la notación, definimos el siguiente conjunto

$$\Delta_\Sigma^{ef} = \{\sigma\}_{\sigma\in\Omega} \cup \{invar_g\}_{g\in G} \cup \{igual_g\}_{g\in G} \cup \{card_g\}_{g\in G^\diamond} \cup \{enum_g\}_{g\in G^\diamond}$$

Para cada G-tipo $\underline{\mathtt{T}}$, denotamos por $\mathcal{M}^{ef,\underline{\mathtt{T}}}$ al siguiente objeto de la categoría $\mathcal{C}Imp_{ef}^{\mathtt{T},\{\}}(\mathcal{T}_{imp})$

- para cada $g \in G$, $\mathcal{M}_g^{ef,\underline{\mathtt{T}}} = \mathtt{T}_g$, $\mathcal{M}_{bool}^{ef,\underline{\mathtt{T}}} = \mathcal{U}$ y $\mathcal{M}_{nat}^{ef,\underline{\mathtt{T}}} = \mathtt{(satisfies\ naturalp)}$;
- para el género distinguido,

  $$\mathcal{M}_{imp_\Sigma}^{ef,\underline{\mathtt{T}}} = \Big\{\overline{f} = (F(\mathtt{p}_\delta))_{\delta\in\Delta_\Sigma^{ef}} \mid \exists\, \mathcal{I} = \big(\underline{\mathcal{R}}, (\mathtt{p}_\delta^{\mathcal{I}})_{\delta\in\Delta_\Sigma^{ef}}\big) \in Obj(Imp_{ef}(\mathcal{T})) \text{ tal que } \underline{\mathtt{T}} \text{ es el } G\text{-tipo de } \underline{\mathcal{R}}\Big\}$$

  siendo, para cada $g \in G^\diamond$, $\mathcal{R}_g$ representación efectiva y, para cada $g \in G \setminus G^\diamond$, $\mathcal{R}_g$ localmente efectiva. Los dominios de las funciones del conjunto anterior son conocidos ya que son funciones que provienen de implementaciones de $Imp_{ef}(\mathcal{T})$. Así, si $\overline{f} \in \mathcal{M}_{imp_\Sigma}^{ef,\underline{\mathtt{T}}}$, existe una implementación de $Imp_{ef}(\mathcal{T})$, $\mathcal{I} = (\underline{\mathcal{R}}, (\mathtt{p}_\delta^{\mathcal{I}})_{\delta\in\Delta_\Sigma^{ef}})$ tal que, para cada $\delta \in \Delta_\Sigma^{ef}$, la función f_δ es la definida por el programa $\mathtt{p}_\delta^{\mathcal{I}}$, siendo los dominios de las funciones de $\overline{f}$ los siguientes:

  · para cada $g \in G$, la función f_{invar_g} es total;
  · para cada $g \in G$, el dominio de la función f_{igual_g} es $\mathcal{S}_g \times \mathcal{S}_g$, donde $\mathcal{S}_g$ denota al soporte de la representación $\mathcal{R}_g$;
  · para cada $g \in G^\diamond$, la función f_{card_g} es constante;
  · para cada $g \in G^\diamond$, el dominio de la función f_{enum_g} es el conjunto $\big\{\mathtt{n} \in \mathtt{(satisfies\ naturalp)} \mid \mathtt{n} < f_{card_g}()\big\}$; y,
  · para cada $(\sigma\colon \omega \to v) \in \Omega$ con $\omega = g_1 \ldots g_n$, el dominio de la función f_σ es $\mathcal{S}_{g_1} \times \ldots \times \mathcal{S}_{g_n}$

- cada operación de $\mathcal{M}^{ef,\underline{\mathtt{T}}}$ se define como la aplicación de la proyección de la tupla de funciones en la componente correspondiente.

El objeto $\mathcal{M}^{ef,\underline{\mathtt{T}}}$ es un objeto especial en la categoría $\mathcal{C}Imp_{ef}^{\mathtt{T},\{\}}(\mathcal{T}_{imp})$, posee las propiedades recogidas en los siguientes teoremas.

Teorema 4.4.5 *El objeto $\mathcal{M}^{ef,\underline{\mathtt{T}}}$ es el coproducto en $\mathcal{C}Imp_{ef}^{\mathtt{T},\{\}}(\mathcal{T}_{imp})$ de los objetos de la imagen de la inclusión de $Imp_{ef}^{\mathtt{T},\{\}}(\mathcal{T})$ en $\mathcal{C}Imp_{ef}^{\mathtt{T},\{\}}(\mathcal{T}_{imp})$.*

Al igual que en los casos anteriores, observar que la inclusión no es inyectiva, puesto que la parte de implementación no recogida en la $\Sigma_{imp_{ef}}$-álgebra que define cada implementación, hace que no lo sea.

Teorema 4.4.6 *La $\Sigma_{imp_{ef}}$-álgebra $\mathcal{M}^{ef,\underline{\mathtt{T}}}$ es final en la categoría $\mathcal{C}Imp_{ef}^{\mathtt{T},\{\}}(\mathcal{T}_{imp})$.*

Denotamos por $\mathcal{C}Imp^{\{\}}_{ef}(\mathcal{T}_{imp})$ a la subcategoría de $\mathcal{C}Imp_{ef}(\mathcal{T}_{imp})$ con los mismos objetos y morfismos aquellos que son la identidad sobre todos los conjuntos soporte excepto el correspondiente al género distinguido imp_Σ. En ella, se tiene el resultado recogido en el siguiente teorema.

Teorema 4.4.7 *La familia* $\{\mathcal{M}^{ef,\underline{\mathtt{T}}}\}_{\underline{\mathtt{T}} \text{ es } G-\text{tipo}}$ *es final en la categoría* $\mathcal{C}Imp^{\{\}}_{ef}(\mathcal{T}_{imp})$.

Así, para recoger cualquier implementación sobre representaciones efectivas del TAD $\mathcal{T}$, las propiedades anteriores permiten que nos limitemos a implementar simplemente las $\Sigma_{imp_{ef}}$-álgebras $\mathcal{M}^{ef,\underline{\mathtt{T}}}$.

4.4.3 Implementaciones canónicas

En esta sección tratamos de implementar cada $\Sigma_{imp_{ef}}$-álgebra $\mathcal{M}^{ef,\underline{\mathtt{T}}}$ de forma canónica, en el mismo sentido que se ha hecho en los casos anteriores, es decir, dando implementaciones finales en ciertas categorías de implementaciones.

La implementación canónica $\mathcal{I}^{ef,\underline{\mathtt{T}},can}$ se define de modo análogo a lo visto en los casos anteriores, es decir, la representación en la máquina de cada implementación del TAD $\mathcal{T}$ sobre representaciones efectivas de algunos conjuntos soporte está formada por una tupla de objetos funcionales que provienen de los programas de las implementaciones de $\mathcal{T}$: algunos de los que implementan a las operaciones del TAD, el resto de las representaciones, efectivas o no. Por tanto, respecto a lo visto, el tratamiento va a ser el mismo, simplemente debemos añadir al tipo que las recoge, por cada género de $G^\diamond$, dos campos en los que almacenar los objetos funcionales que implementan a las enumeraciones. Suponemos que en la enumeración de los géneros de Σ los l primeros son los géneros de $G^\diamond$. Así, lo anterior se traduce en que el tipo que en este caso utilizamos para el género imp_Σ es el siguiente

$$\texttt{(defstruct REP-ef-IMP-g imp- rep-card}_{g_1} \ \ldots \ \texttt{rep-card}_{g_l} \ \texttt{rep-enum}_{g_1} \ \ldots \ \texttt{rep-enum}_{g_l} \ \texttt{(:include REP-IMP-g))}$$

siendo los datos del dominio los del tipo anterior cuyos campos son todos objetos funcionales.

Como en todos los casos recogidos anteriormente, el punto más delicado es la definición del soporte de la representación del género distinguido. Informalmente, los datos del soporte son aquellas tuplas de objetos funcionales que provienen de implementaciones de la categoría $Imp^{\underline{\mathtt{T}}}_{ef}(\mathcal{T})$. Por tanto, si `x` está en el dominio, `x` es un dato del soporte de la representación para imp_Σ si existe $\mathcal{J} = \left(\underline{\mathcal{R}}^{\mathcal{J}}, (\mathtt{p}^{\mathcal{J}}_\delta)_{\delta \in \Delta^{ef}_\Sigma}\right) \in Obj(Imp^{\underline{\mathtt{T}}}_{ef}(\mathcal{T}))$ tal que los códigos de los programas de $\mathcal{J}$ coinciden (por la igualdad `eq`, única razonable entre objetos funcionales) con los almacenados en los campos de `x`. Formalmente, debe verificarse:

- $\texttt{(eq c}^{\mathcal{J}}_{invar_g} \texttt{ (REP-ef-IMP-g-rep-invar}_g \texttt{ x))}$, para cada $g \in G$, siendo $\mathtt{c}^{\mathcal{J}}_{invar_g}$ el código del programa que implementa al invariante; esto viene a asegurar que cada soporte $\mathcal{S}^{\mathcal{J}}_g$ coincide con el correspondiente subconjunto $S^{\mathtt{x}}_g$ de $\mathtt{T}_g$ siendo

$$S^{\mathtt{x}}_g = \left\{\mathtt{d} \in \mathtt{T}_g \mid \texttt{(not (eq (funcall (REP-ef-IMG-g-rep-invar}_g \texttt{ x) d) nil))}\right\}$$

- `(eq` $\mathtt{c}^{\mathcal{J}}_{igual_g}$ `(REP-ef-IMP-g-rep-igual`$_g$ `x))`, también para cada $g \in G$, y donde $\mathtt{c}^{\mathcal{J}}_{igual_g}$ es el código del programa que implementa a la igualdad de la representación y que tiene por soporte $\mathcal{S}^{\mathcal{J}}_g \times \mathcal{S}^{\mathcal{J}}_g$;

- `(eq` $\mathtt{c}^{\mathcal{J}}_{card_g}$ `(REP-ef-IMP-g-rep-card`$_g$ `x))`, para cada $g \in G^{\diamond}$, y donde $\mathtt{c}^{\mathcal{J}}_{card_g}$ es el código del programa $\mathtt{p}^{\mathcal{J}}_{card_g} = (\mathtt{c}^{\mathcal{J}}_{card_g},\ \mathtt{nil},\ \mathtt{nil},\ \mathtt{(satisfies\ naturalp)})$;

- `(eq` $\mathtt{c}^{\mathcal{J}}_{enum_g}$ `(REP-ef-IMP-g-rep-enum`$_g$ `x))`, también para cada $g \in G^{\diamond}$, y donde $\mathtt{c}^{\mathcal{J}}_{enum_g}$ es el código del programa $\mathtt{p}^{\mathcal{J}}_{enum_g} = \big(\mathtt{c}^{\mathcal{J}}_{enum_g},\ \{\mathtt{n} \in \mathtt{(satisfies\ naturalp)} \mid \mathtt{(<\ n}\ F(\mathtt{p}^{\mathcal{J}}_{card_g})()\mathtt{)}\}\big)$, `(satisfies naturalp)`, $\mathtt{T}_g\big)$; además, para cada $g \in G^{\diamond}$, $F(\mathtt{p}^{\mathcal{J}}_{card_g})$ y $F(\mathtt{p}^{\mathcal{J}}_{enum_g})$ verifican las condiciones correspondientes a las enumeraciones;

- `(eq` $\mathtt{c}^{\mathcal{J}}_{\sigma}$ `(REP-ef-IMP-g-imp-`σ `x))`, para cada $(\sigma\colon g_1 \dots g_k \to g) \in \Omega$, siendo $\mathtt{p}^{\mathcal{J}}_{\sigma} = (\mathtt{c}^{\mathcal{J}}_{\sigma},\ \mathcal{S}^{\mathcal{J}}_{g_1} \times \cdots \times \mathcal{S}^{\mathcal{J}}_{g_k},\ \mathtt{T}_{g_1},\ \dots,\ \mathtt{T}_{g_k},\ \mathtt{T}_g)$ el programa que implementa al operador σ en la implementación $\mathcal{J}$.

La función de abstracción lleva cada tupla de objetos funcionales a la tupla de funciones definidas por esos códigos sobre los dominios deducidos de los de la implementación de $Imp^{\underline{\mathtt{T}}}_{ef}(\mathcal{T})$ que hace que la tupla de códigos esté en el soporte.

Los programas que implementan a las operaciones que provienen de $\Omega_{imp_{rep}}$ son los mismos que en el caso de la implementación $\mathcal{I}^{rep,\underline{\mathtt{T}},can}$. Los programas que implementan a las enumeraciones, son, para cada $g \in G^{\diamond}$, los siguientes:

- el programa $\mathtt{p}^{ef,\underline{\mathtt{T}},can}_{rep\_card_g}$ que implementa a la función $rep\_card_{\mathcal{M}^{ef,\underline{\mathtt{T}}}_g}\colon \mathcal{M}^{ef,\underline{\mathtt{T}}}_{imp_\Sigma} \to$ (`satisfies naturalp`) tiene por soporte $\mathcal{S}^{ef,\underline{\mathtt{T}},can}_{imp_\Sigma}$ y por código el siguiente objeto funcional

$$\mathtt{c}^{\mathtt{p}^{ef,\underline{\mathtt{T}},can}_{rep\_card_g}} \equiv \mathtt{\#'(lambda\ (x)\ (funcall\ (REP\text{-}ef\text{-}IMP\text{-}g\text{-}rep\text{-}card}_g\ \mathtt{x)))}$$

- la función $rep\_enum_{\mathcal{M}^{ef,\underline{\mathtt{T}}}_g}\colon \mathcal{M}^{ef,\underline{\mathtt{T}}}_{imp_\Sigma} \times$ (`satisfies naturalp`) $\to \mathtt{T}_g$ es implementada por el programa $\mathtt{p}^{ef,\underline{\mathtt{T}},can}_{rep\_enum_g}$ con tipos de entrada $\mathcal{D}^{ef,\underline{\mathtt{T}},can}_{imp_\Sigma}$ y `(satisfies naturalp)`, de salida $\mathtt{T}_g$, soporte el conjunto $\big\{(\mathtt{x},\mathtt{n}) \mid \mathtt{n} < F(\mathtt{p}^{ef,\underline{\mathtt{T}},can}_{rep\_card_g})(\mathtt{x})\big\}$, y código el siguiente objeto funcional

$$\mathtt{c}^{\mathtt{p}^{ef,\underline{\mathtt{T}},can}_{rep\_enum_g}} \equiv \mathtt{\#'(lambda\ (x\ n)\ (funcall\ (REP\text{-}ef\text{-}IMP\text{-}g\text{-}rep\text{-}enum}_g\ \mathtt{x)\ n))}$$

Análogamente a como se ha hecho en los casos anteriores se obtiene el resultado recogido en la siguiente proposición.

Proposición 4.4.8 *Cada $\mathcal{I}^{ef,\underline{\mathtt{T}},can}$ es una implementación de la $\Sigma_{imp_{ef}}$-álgebra $\mathcal{M}^{ef,\underline{\mathtt{T}}}$.*

4.4.4 Categorías de implementaciones sobre representaciones efectivas

Nos centramos a continuación en definir las subcategorías de implementaciones de $\mathcal{M}^{ef,\underline{\mathtt{T}}}$ en las que las implementaciones $\mathcal{I}^{ef,\underline{\mathtt{T}},can}$ serán objetos finales. Para ello, lo primero que observamos

es que las propiedades que verifica cada $\mathcal{I}^{ef,\underline{\mathtt{T}},can}$ respecto a las operaciones que están en $\Omega_{imp_{rep}}$ son las mismas que verifica la implementación $\mathcal{I}^{rep,\underline{\mathtt{T}},can}$ y que están recogidas en la página 224 de esta memoria. Respecto a las que no provienen de $\Omega_{imp_{rep}}$, las relativas a las enumeraciones, $\mathcal{I}^{ef,\underline{\mathtt{T}},can}$ verifica las siguientes propiedades:

- cada `x` del soporte del género imp_Σ determina una constante para cada $g \in G^\diamond$, `(funcall (REP-ef-IMP-g-rep-card-g x))`, que se utiliza para definir el dominio de la función $\mathtt{f}^{\mathtt{x}}_{enum_g}$ que está en la imagen de `x` por la abstracción, es decir, $Def(\mathtt{f}^{\mathtt{x}}_{enum_g}) = \{\mathtt{n} \in$ `(satisfies naturalp)` $\mid$ `(< n (funcall (REP-ef-IMP-g-rep-card-g x)))`$\}$
- para cada $g \in G^\diamond$, el dominio del programa que implementa a $rep\_enum_{\mathcal{M}^{ef,\underline{\mathtt{T}}}_g}$ puede expresarse como

$$\bigcup_{\mathtt{x} \in \mathcal{S}^{ef,\underline{\mathtt{T}},can}_{imp_\Sigma}} (\{\mathtt{x}\} \times Def(\mathtt{f}^{\mathtt{x}}_{enum_g}))$$

Apoyándonos en las propiedades verificadas por cada implementación $\mathcal{I}^{ef,\underline{\mathtt{T}},can}$, denotamos por $Imp^{\mathtt{T},\{\}}_{ef}(\mathcal{M}^{ef,\underline{\mathtt{T}}})$ a la subcategoría de la categoría de implementaciones de $\mathcal{M}^{ef,\underline{\mathtt{T}}}$ con objetos las implementaciones $\mathcal{I} = (\underline{\mathcal{R}}, (\mathtt{p}^{\mathcal{I}}_{rep\_invar_g})_{g\in G}, (\mathtt{p}^{\mathcal{I}}_{rep\_igual_g})_{g\in G}, (\mathtt{p}^{\mathcal{I}}_{rep\_card_g})_{g\in G^\diamond}, (\mathtt{p}^{\mathcal{I}}_{rep\_enum_g})_{g\in G^\diamond}, (\mathtt{p}^{\mathcal{I}}_{imp\_\sigma})_{\sigma\in\Omega})$, que verifican las siguientes condiciones:

i) Todas las condiciones que verifican las implementaciones de la categoría $Imp^{\mathtt{T},\{\}}_{rep}(\mathcal{M}^{rep,\underline{\mathtt{T}}})$, recogidas en los apartados *i*), *ii*) y *iii*) de la página 224 de esta memoria.

ii) Para cada $g \in G^\diamond$, el dominio del programa $\mathtt{p}^{\mathcal{I}}_{rep\_enum_g}$ es

$$Def(\mathtt{p}^{\mathcal{I}}_{rep\_enum_g}) = \{(\mathtt{x},\mathtt{n}) \in \mathcal{S}^{\mathcal{I}}_{imp_\Sigma} \times \texttt{(satisfies naturalp)} \mid \mathtt{n} < F(\mathtt{p}^{\mathcal{I}}_{rep\_card_g})()\}$$

Como morfismos de la categoría, se toman las i–transformaciones que entre las representaciones efectivas son r–transformaciones, siendo la identidad sobre todos los modelos y sobre los soportes de todas las representaciones excepto la del género imp_Σ.

A partir de cada implementación $\mathcal{I}$ de la categoría $Imp^{\mathtt{T}}_{ef}(\mathcal{T})$ podemos definir una implementación de $\mathcal{M}^{ef,\underline{\mathtt{T}}}$, implementación muy parcial en el sentido de que solo alcanza la tupla de objetos funcionales que provienen de $\mathcal{I}$. Esta implementación de $\mathcal{M}^{ef,\underline{\mathtt{T}}}$ a la que denotamos por $\mathcal{I}^*$ se define restringiendo el soporte del género distinguido a una única tupla de objetos funcionales, los que provienen de $\mathcal{I}$. Así, podemos asociar a cada implementación $\mathcal{I}$ de $Imp^{\mathtt{T}}_{ef}(\mathcal{T})$ una implementación $\mathcal{I}^*$ del modelo $\mathcal{M}^{ef,\underline{\mathtt{T}}}$, construcción que se extiende a un funtor (inclusión) entre las correspondientes categorías y que nos lleva al resultado recogido en el siguiente teorema.

Teorema 4.4.9 *La implementación $\mathcal{I}^{ef,\underline{\mathtt{T}},can}$ es el coproducto en $Imp^{\mathtt{T},\{\}}_{ef}(\mathcal{M}^{ef,\underline{\mathtt{T}}})$ de las implementaciones que forman la imagen del funtor inclusión de la categoría $Imp^{\mathtt{T},\{\}}_{ef}(\mathcal{T})$ en $Imp^{\mathtt{T},\{\}}_{ef}(\mathcal{M}^{ef,\underline{\mathtt{T}}})$.*

Otra propiedad de las implementaciones canónicas es la recogida a continuación.

Teorema 4.4.10 *La implementación canónica $\mathcal{I}^{ef,\underline{\mathtt{T}},can}$ es objeto final en la categoría $Imp_{ef}^{\underline{\mathtt{T}},\{\}}(\mathcal{M}^{ef,\underline{\mathtt{T}}})$.*

Denotamos por $\mathcal{T}_{imp}$ al TAD formado por todos los modelos $\mathcal{M}^{ef,\underline{\mathtt{T}}}$, para cada G-tipo $\underline{\mathtt{T}}$, y por $Imp_{ef}^{\{\}}(\mathcal{T}_{imp})$ a la categoría de implementaciones obtenida por agrupación de las categorías $Imp_{ef}^{\underline{\mathtt{T}},\{\}}(\mathcal{M}^{ef,\underline{\mathtt{T}}})$. El siguiente resultado explica cómo se agrupan todos estos objetos finales.

Corolario 4.4.11 *La familia de implementaciones canónicas $\{\mathcal{I}^{ef,\underline{\mathtt{T}},can}\}_{\underline{\mathtt{T}} \text{ es } G-\text{tipo}}$ es una familia final en la categoría $Imp_{ef}^{\{\}}(\mathcal{T}_{imp})$.*

4.4.5 Aplicación al Cálculo Simbólico: Complejos de cadenas

Volvemos al caso de los complejos de cadenas y concretamente a la $\mathtt{CC}_{imp_{rep}}$-álgebra $\mathcal{M}^{rep,\underline{\mathtt{T}},red,gen}$ definida en la página 235, cuya descripción pretende establecer la mínima información imprescindible para recuperar un complejo de cadenas a partir de una implementación de $Imp_{Dec}^{\underline{\mathtt{T}}^{nat}}(\mathcal{T}_{\mathtt{CC}})$ (implementaciones de complejos de cadenas con conjunto soporte para *cmbn* un subconjunto de *Cmb*, sobre representaciones naturales y con invariantes e igualdades de representación calculables y explícitos). El conjunto soporte para el género $imp_{\mathtt{CC}}$ corresponde a $\mathcal{M}_{imp_{\mathtt{CC}}}^{rep,\underline{\mathtt{T}},red,gen}$. La idea es que cada implementación $\mathcal{I}$ de $Imp_{Dec}^{\underline{\mathtt{T}}^{nat}}(\mathcal{T}_{\mathtt{CC}})$, queda suficientemente determinada por una terna de funciones $\big(\mathtt{f}_{invar_{gen}}\colon \mathtt{integer} \times \mathcal{U} \to \mathcal{U}$, $\mathtt{f}_{igual_{gen}}\colon \mathtt{integer} \times \mathcal{U} \times \mathcal{U} \to \mathcal{U}$, $\mathtt{f}_{dffr}\colon \mathtt{T}_{cmbn} \to \mathtt{T}_{cmbn}\big)$ que definen: el conjunto de generadores en cada grado, una relación de igualdad entre ellos y el comportamiento de la diferencial del complejo de cadenas. Para llegar a esta expresión, el punto de partida es un conjunto graduado $(\mathtt{G}, =_{\mathtt{G}})$ que determine los generadores del complejo de cadenas, que vendrá dado por medio de un PER graduado en $\mathcal{U}$. Vimos que una forma de implementar el PER graduado $(\mathtt{G}, =_{\mathtt{G}})$ es por medio de un par de funciones $(\mathtt{invar}_{\mathtt{G}}\colon \mathtt{integer} \times \mathcal{U} \to \mathcal{U}$, $\mathtt{igual}_{\mathtt{G}}\colon \mathtt{integer} \times \mathcal{U} \times \mathcal{U} \to \mathcal{U})$. La función $\mathtt{invar}_{\mathtt{G}}\colon \mathtt{integer} \times \mathcal{U} \to \mathcal{U}$ es total y determina el conjunto de generadores en cada grado, así $\mathtt{invar}_{\mathtt{G}}(\mathtt{p},\mathtt{g}) \neq_{\mathtt{eq}} \mathtt{nil}$ si y solo si $\mathtt{g} \in \mathtt{G}_{\mathtt{p}}$. La función $\mathtt{igual}_{\mathtt{G}}\colon \mathtt{integer} \times \mathcal{U} \times \mathcal{U} \to \mathcal{U}$ es una función parcial con dominio $\big\{(\mathtt{p},\mathtt{g1},\mathtt{g2}) \mid \mathtt{invar}_{\mathtt{G}}(\mathtt{p},\mathtt{g1}) \neq_{\mathtt{eq}} \mathtt{nil}$ e $\mathtt{invar}_{\mathtt{G}}(\mathtt{p},\mathtt{g2}) \neq_{\mathtt{eq}} \mathtt{nil}\big\}$, y con comportamiento $\mathtt{igual}_{\mathtt{G}}(\mathtt{p},\mathtt{g1},\mathtt{g2}) := \mathtt{g1} =_{\mathtt{G}_{\mathtt{p}}} \mathtt{g2}$.

En el caso de que el PER graduado $(\mathtt{G}, =_{\mathtt{G}})$ sea finito, es decir, todos los grupos del complejo de cadenas sean finitamente generados, la implementación anterior es parcial en el sentido de que pierde parte de la información que puede deducirse de $(\mathtt{G}, =_{\mathtt{G}})$. Desde el punto de vista matemático, al considerar $(\mathtt{G}, =_{\mathtt{G}})$ realmente estamos trabajando en el cociente $\mathtt{G}/=_{\mathtt{G}}$ y, siendo que para cada $\mathtt{p} \in \mathtt{integer}$, $\mathtt{G}_{\mathtt{p}}$ es finito, es posible conocer y tener explícitamente un conjunto de representantes canónicos del cociente anterior. Es decir, para cada $\mathtt{p} \in \mathtt{integer}$, se tiene un conjunto $\mathtt{Can}_{\mathtt{p}} \subseteq \mathtt{G}_{\mathtt{p}}$ verificando las siguientes propiedades:

i) si $\mathtt{g}, \mathtt{g}' \in \mathtt{Can}_{\mathtt{p}}$, entonces $\mathtt{g} \neq_{\mathtt{G}_{\mathtt{p}}} \mathtt{g}'$

ii) para cada $\mathtt{g} \in \mathtt{G}_{\mathtt{p}} \setminus \mathtt{Can}_{\mathtt{p}}$, existe $\mathtt{g}_{\mathtt{c}} \in \mathtt{Can}_{\mathtt{p}}$ tal que $\mathtt{g} =_{\mathtt{G}_{\mathtt{p}}} \mathtt{g}_{\mathtt{c}}$

En este caso, se puede definir una enumeración sobre $\mathtt{Can}_{\mathtt{G}} = (\mathtt{Can}_{\mathtt{p}})_{\mathtt{p} \in \mathtt{integer}}$, enumeración que puede venir dada por un par de funciones $\mathtt{card}_{\mathtt{G}}\colon \mathtt{integer} \to (\mathtt{satisfies\ naturalp})$ y $\mathtt{enum}_{\mathtt{G}}\colon \mathtt{integer} \times (\mathtt{satisfies\ naturalp}) \to \mathcal{U}$, siendo $\mathtt{card}_{\mathtt{G}}$ una función total que determina el cardinal del conjunto $\mathtt{Can}_{\mathtt{p}}$, para cada $\mathtt{p} \in \mathtt{integer}$, y $\mathtt{enum}_{\mathtt{G}}$ función parcial con dominio

$\{(\mathtt{p},\mathtt{n}) \in \mathtt{integer} \times (\mathtt{satisfies\ naturalp}) \mid \mathtt{n} < \mathtt{card_G}(\mathtt{p})\}$ y de forma que $\mathtt{enum_G}$ determina una enumeración del conjunto $\mathtt{Can_p}$, para cada $\mathtt{p} \in \mathtt{integer}$.

Una forma de introducir esta información en la máquina es considerando una representación efectiva para el género correspondiente. La signatura CC posee como géneros *int* y *cmbn* y para recoger la información anterior tomamos representaciones efectivas para el conjunto soporte del género *cmbn*. Por contra, para el género *int* venimos considerando representaciones naturales (que siempre serán de naturaleza localmente efectiva).

Así, partimos del tipo $\mathtt{T}_{cmbn}$ y de su representación natural. El invariante $\mathtt{invar_G}$ (que en este caso no corresponde con el invariante de la representación sino con el invariante de los generadores de los grupos del complejo de cadenas) nos permite determinar los dominios de definición de los programas. Por ello, para recoger la información matemática deducida a partir del PER graduado $(\mathtt{G}, =_{\mathtt{G}})$, no debemos dar enumeración ni del soporte ni de su cociente por la igualdad de la representación, sino que lo que interesa enumerar es el cociente del conjunto graduado de generadores por la relación de igualdad. Es decir, vamos a enumerar $(\mathtt{G}, =_{\mathtt{G}})$ y no $(\mathcal{S}_{cmbn}, =_{\mathcal{R}_{cmbn}})$. Por ello, el caso de enumeración que recogemos en este ejemplo, no corresponde al caso desarrollado en la parte teórica anterior, como ya anunciamos, sino que es más rico en el sentido de que se enumeran generadores, en este caso, con la noción de generabilidad de las combinaciones de un complejo de cadenas por linealidad, a partir de un conjunto mínimo de datos.

Así, si $(\mathtt{G}, =_{\mathtt{G}})$ es un PER graduado en $\mathcal{U}$, una enumeración del PER anterior viene dada por un par de funciones $\big(\mathtt{card_G}\colon \mathtt{integer} \to (\mathtt{satisfies\ naturalp}),\ \mathtt{enum_G}\colon \mathtt{integer} \times (\mathtt{satisfies\ naturalp}) \to \mathcal{U}\big)$ siendo $\mathtt{card_G}\colon \mathtt{integer} \to (\mathtt{satisfies\ naturalp})$ una función total definida por $\mathtt{card_G}(\mathtt{p}) := card(\mathtt{Can_p})$, y $\mathtt{enum_G}\colon \mathtt{integer} \times (\mathtt{satisfies\ naturalp}) \to \mathcal{U}$ una función parcial con dominio $\{(\mathtt{p},\mathtt{n}) \mid \mathtt{n} < card(\mathtt{Can_p})\}$ y definida de forma que $\mathtt{enum_G}(\mathtt{p},\mathtt{n})$ es el representante canónico de $\mathtt{G_p}$ que ocupa la posición $n-1$ en la enumeración de los elementos del conjunto $\mathtt{Can_p}$, teniendo en cuenta que para realmente tener una enumeración se deben verificar las siguientes condiciones:

i) para cada par $(\mathtt{p},\mathtt{n}) \in Def(\mathtt{enum_G})$, se tiene que $\mathtt{invar_G}\big(\mathtt{p}, \mathtt{enum_G}(\mathtt{p},\mathtt{n})\big) \neq_{\mathtt{eq}} \mathtt{nil}$

ii) si $(\mathtt{p},\mathtt{n1}), (\mathtt{p},\mathtt{n2}) \in Def(\mathtt{enum_G})$, entonces $\mathtt{igual_G}\big(\mathtt{p}, \mathtt{enum_G}(\mathtt{p},\mathtt{n1}), \mathtt{enum_G}(\mathtt{p},\mathtt{n2})\big) =_{\mathtt{eq}} \mathtt{nil}$

iii) para cada par $(\mathtt{p},\mathtt{g}) \in \mathtt{integer} \times \mathcal{U}$ tal que $\mathtt{invar_G}(\mathtt{p},\mathtt{g}) \neq_{\mathtt{eq}} \mathtt{nil}$, se tiene que existe $\mathtt{n} \in$ (satisfies naturalp) con $(\mathtt{p},\mathtt{n}) \in Def(\mathtt{enum_G})$ y tal que $\mathtt{igual_G}\big(\mathtt{p},\mathtt{g}, \mathtt{enum_G}(\mathtt{p},\mathtt{n})\big) \neq_{\mathtt{eq}}$ nil

Lo anterior nos lleva a trabajar con una subcategoría propia de implementaciones definida a partir de $Imp^{\mathtt{T}^{nat}}_{Dec}(\mathcal{T}_{\mathtt{CC}})$. Denotamos por $Imp^{\mathtt{T}^{nat}}_{ef}(\mathcal{T}_{\mathtt{CC}})$ a la subcategoría de implementaciones del TAD $\mathcal{T}_{\mathtt{CC}}$ con objetos las implementaciones con G-tipo (integer, $\mathtt{T}_{cmbn}$) sobre representaciones efectivas para el género *cmbn*, de forma que olvidando la parte efectiva, se tiene una implementación de $Imp^{\mathtt{T}^{nat}}_{Dec}(\mathcal{T}_{\mathtt{CC}})$. Es decir, siendo precisos, estamos con implementaciones de $\mathcal{T}_{\mathtt{CC}}$ con G-tipo (integer, $\mathtt{T}_{cmbn}$) de forma que la representación para el género *int* es la natural, la del género *cmbn* es natural y efectiva, ambas explícitas y los dominios de los programas que implementan a las operaciones *opp*, *zero*, *mult* y *dffr* son los productos de los soportes de las representaciones correspondientes, siendo el dominio del que implementa a la operación *suma* el formado por los pares de datos del soporte de la representación del mismo grado. Observar que las implementaciones de $Imp^{\mathtt{T}^{nat}}_{ef}(\mathcal{T}_{\mathtt{CC}})$ son implementaciones de complejos de cadenas

finitamente generados. (Realmente en este caso podíamos haber partido de los complejos de cadenas finitamente generados y considerar implementaciones sobre la representación natural de `integer` y la representación natural y efectiva para el tipo $\mathtt{T}_{cmbn}$ que verifiquen las condiciones sobre los dominios de definición de los programas. Hemos preferido hacerlo así para que quede más claro que un objeto efectivo es, en particular, localmente efectivo.)

Cada implementación de la categoría anterior puede verse como una estructura

$$\mathcal{I} = \Big(\underline{\mathcal{R}},\, (\mathtt{p}^{\mathcal{I}}_{\sigma})_{\sigma\in\{suma,opp,zero,mult,dffr\}},\, (\mathtt{p}^{\mathcal{I}}_{invar_{int}},\, \mathtt{p}^{\mathcal{I}}_{invar_{cmbn}}),\, (\mathtt{p}^{\mathcal{I}}_{igual_{int}},\, \mathtt{p}^{\mathcal{I}}_{igual_{cmbn}}),\, \mathtt{p}^{\mathcal{I}}_{card_{cmbn}},\, \mathtt{p}^{\mathcal{I}}_{enum_{cmbn}}\Big)$$

siendo: $\mathtt{p}^{\mathcal{I}}_{invar_g}$ un programa que implementa al invariante de la representación $\mathcal{R}_g$ con $g \in \{int, cmbn\}$, $\mathtt{p}^{\mathcal{I}}_{igual_g}$ un programa que implementa a la igualdad de la representación $\mathcal{R}_g$ con $g \in \{int, cmbn\}$ y $(\mathtt{p}^{\mathcal{I}}_{card_{cmbn}},\, \mathtt{p}^{\mathcal{I}}_{enum_{cmbn}})$ una enumeración de los representantes canónicos de los generadores del complejo de cadenas implementado. Así, si $\mathcal{I}$ es una implementación de un complejo de cadenas con generadores definidos por el PER $(\mathtt{G}, =_{\mathtt{G}})$, $\big(\mathtt{invar_G}\colon \mathtt{integer} \times \mathcal{U} \to \mathcal{U},$ $\mathtt{igual_G}\colon \mathtt{integer} \times \mathcal{U} \times \mathcal{U} \to \mathcal{U}\big)$ es una implementación del PER $(\mathtt{G}, =_{\mathtt{G}})$ y $\big(\mathtt{card_G}\colon \mathtt{integer} \to$ $(\mathtt{satisfies\ naturalp})$, $\mathtt{enum_G}\colon \mathtt{integer} \times (\mathtt{satisfies\ naturalp}) \to \mathcal{U}\big)$ dan una enumeración del conjunto graduado $(\mathtt{Can_p})_{\mathtt{p}\in\mathtt{integer}}$, (para cada $\mathtt{p} \in \mathtt{integer}$, el conjunto $\mathtt{Can_p}$ es el formado por los representantes canónicos del cociente $\mathtt{G_p}/_{=_{\mathtt{G_p}}}$), entonces las funciones definidas por los programas de $\mathcal{I}$ que implementan al invariante y a la igualdad de representación, $F(\mathtt{p}^{\mathcal{I}}_{invar_{cmbn}})$ y $F(\mathtt{p}^{\mathcal{I}}_{igual_{cmbn}})$, respectivamente, se definen a partir de ellas tal y como se ha explicado (en la página 233). La función definida por el programa $\mathtt{p}^{\mathcal{I}}_{card_{cmbn}}$ es total y con comportamiento $F(\mathtt{p}^{\mathcal{I}}_{card_{cmbn}})(\mathtt{p}) := \mathtt{card_G}(\mathtt{p})$. Finalmente, la función definida por el programa $\mathtt{p}^{\mathcal{I}}_{enum_{cmbn}}$ tiene por dominio $\big\{(\mathtt{p},\mathtt{n}) \mid \mathtt{n} < F(\mathtt{p}^{\mathcal{I}}_{card_{cmbn}})(\mathtt{p})\big\}$ y su comportamiento viene dado por $F(\mathtt{p}^{\mathcal{I}}_{enum_{cmbn}})(\mathtt{p},\mathtt{n}) := \mathtt{enum_G}(\mathtt{p},\mathtt{n})$. Por coherencia con la notación empleada hasta ahora, denotaremos por $\mathtt{p}^{\mathcal{I}}_{card_{gen}}$ y $\mathtt{p}^{\mathcal{I}}_{enum_{gen}}$ a los programas que implementan a las enumeraciones, aunque lo son solo de generadores y no de combinaciones. Además, por ser una enumeración, se verifican las siguientes condiciones:

i) para cada par $(\mathtt{p},\mathtt{n}) \in \mathtt{S}^{\mathtt{p}^{\mathcal{I}}_{enum_{gen}}}$, se tiene que

$$F(\mathtt{p}^{\mathcal{I}}_{invar_{cmbn}})\big(\langle \mathtt{p}, [(\mathtt{1}, F(\mathtt{p}^{\mathcal{I}}_{enum_{gen}}(\mathtt{p},\mathtt{n})))]\rangle\big) \neq_{\mathtt{eq}} \mathtt{nil}$$

ii) si $(\mathtt{p},\mathtt{n1}), (\mathtt{p},\mathtt{n2}) \in \mathtt{S}^{\mathtt{p}^{\mathcal{I}}_{enum_{gen}}}$, se tiene que

$$F(\mathtt{p}^{\mathcal{I}}_{igual_{cmbn}})\big(\langle \mathtt{p}, [(\mathtt{1}, F(\mathtt{p}^{\mathcal{I}}_{enum_{gen}}(\mathtt{p},\mathtt{n1})))]\rangle, \langle \mathtt{p}, [(\mathtt{1}, F(\mathtt{p}^{\mathcal{I}}_{enum_{gen}}(\mathtt{p},\mathtt{n2})))]\rangle\big) =_{\mathtt{eq}} \mathtt{nil}$$

iii) para cada $\langle \mathtt{p}, [(\mathtt{1},\mathtt{g})]\rangle \in \mathtt{T}_{cmbn}$ tal que $F(\mathtt{p}^{\mathcal{I}}_{invar_{cmbn}})\big(\langle \mathtt{p}, [(\mathtt{1},\mathtt{g})]\rangle\big) \neq_{\mathtt{eq}} \mathtt{nil}$, se tiene que existe $\mathtt{n} \in \mathtt{(satisfies\ naturalp)}$ con $(\mathtt{p},\mathtt{n}) \in \mathtt{S}^{\mathtt{p}^{\mathcal{I}}_{enum_{gen}}}$ y tal que

$$F(\mathtt{p}^{\mathcal{I}}_{igual_{cmbn}})\big(\langle \mathtt{p}, [(\mathtt{1},\mathtt{g})]\rangle, \langle \mathtt{p}, [(\mathtt{1}, F(\mathtt{p}^{\mathcal{I}}_{enum_{gen}}(\mathtt{p},\mathtt{n})))]\rangle\big) \neq_{\mathtt{eq}} \mathtt{nil}$$

A partir de la categoría $Imp^{\mathtt{T}^{nat}}_{ef}(\mathcal{T}_{\mathtt{CC}})$ definimos una $\mathtt{CC}_{imp_{ef}}$-álgebra $\mathcal{M}^{ef,\mathtt{T}}$, con conjunto soporte para el género $imp_{\mathtt{CC}}$ el espacio de funciones en el que cada elemento es una tupla

formada por todas las funciones que provienen de una implementación de $Imp_{ef}^{\mathtt{T}^{nat}}(\mathcal{T}_{\mathtt{CC}})$, es decir, cada dato es una tupla de funciones $(\mathtt{f}_\delta)_{\delta\in\Delta^{ef}_{\mathtt{CC}}}$, álgebra final en la categoría de álgebras que recogen a las familias de implementaciones de $Imp_{ef}^{\mathtt{T}^{nat}}(\mathcal{T}_{\mathtt{CC}})$.

Sin embargo, lo interesante en la práctica es que podemos aplicar varias simplificaciones en la descripción del conjunto soporte para el género $imp_{\mathtt{CC}}$, similares a las realizadas en el caso localmente efectivo y que aquí se aplican del mismo modo. Concretamente, se tiene que el comportamiento de los programas que implementan a las operaciones *opp*, *zero* y *mult* de cualquier implementación de $Imp_{ef}^{\mathtt{T}^{nat}}(\mathcal{T}_{\mathtt{CC}})$ está determinado por el de las funciones $\mathtt{opp}^{\mathcal{U}}$, $\mathtt{zero}^{\mathcal{U}}$ y $\mathtt{mult}^{\mathcal{U}}$ (descritas en la página 197) considerando dominios de definición adecuados (en este caso definidos a partir de los invariantes, tal y como se ha hecho en el caso localmente efectivo; ver página 234 y siguientes). Respecto de la operación *suma*, su dominio de definición está determinado por el de la función $\mathtt{suma}^{\mathcal{U},\mathtt{igual}^{\mathcal{I}}_{gen}}$ (descrita en la página 234) siendo $\mathtt{igual}^{\mathcal{I}}_{gen}$ la función que define la igualdad sobre los generadores en la implementación $\mathcal{I}$. Esto hace que de la descripción general del objeto final, puedan eliminarse las componentes correspondientes a estas operaciones. Además, del mismo modo que vimos en el caso localmente efectivo, el invariante y la igualdad de la representación para el género *int* en cualquier implementación de $Imp_{ef}^{\mathtt{T}^{nat}}(\mathcal{T}_{\mathtt{CC}})$ vienen dados por `integerp` e `=`, respectivamente, por lo que tampoco es necesario almacenarlos explícitamente como componentes del objeto final. Esto nos permite, pensando en las operaciones de la signatura `CC` y las correspondientes a invariantes e igualdades, partir de la descripción del objeto $\mathcal{M}^{rep,\mathtt{T},red,gen}$ (dada en la página 235, para el caso localmente efectivo) y completarla con la enumeración del género *cmbn*, obteniendo así una descripción más simplificada del álgebra final.

Así, una descripción más reducida de la $\mathtt{CC}_{imp_{ef}}$-álgebra $\mathcal{M}^{ef,\mathtt{T}}$, que denotamos por $\mathcal{M}^{ef,\mathtt{T},red,gen}$ y en la que se recoge la mínima información imprescindible para recuperar un álgebra definida a partir de una implementación de $Imp_{ef}^{\mathtt{T}^{nat}}(\mathcal{T}_{\mathtt{CC}})$, tiene el siguiente conjunto como soporte para el género $imp_{\mathtt{CC}}$

$$\mathcal{M}^{ef,\mathtt{T},red,gen}_{imp_{\mathtt{CC}}} = \Big\{(\mathtt{f}_{invar_{gen}},\ \mathtt{f}_{igual_{gen}},\ \mathtt{f}_{card_{gen}},\ \mathtt{f}_{enum_{gen}},\ \mathtt{f}_{dffr}) \ \mid\ \exists\ \mathcal{I}\in Obj(Imp_{ef}^{\mathtt{T}^{nat}}(\mathcal{T}_{\mathtt{CC}})) \text{ verificando que } \mathtt{f}_{invar_{cmbn}} = F(\mathtt{p}^{\mathcal{I}}_{invar_{cmbn}}),\ \mathtt{f}_{igual_{cmbn}} = F(\mathtt{p}^{\mathcal{I}}_{igual_{cmbn}}),\ \mathtt{f}_{card_{gen}} = F(\mathtt{p}^{\mathcal{I}}_{card_{gen}}),\ \mathtt{f}_{enum_{gen}} = F(\mathtt{p}^{\mathcal{I}}_{enum_{gen}}) \text{ y } \mathtt{f}_{dffr} = F(\mathtt{p}^{\mathcal{I}}_{dffr})\Big\}$$

siendo $\mathtt{f}_{invar_{cmbn}}$ y $\mathtt{f}_{igual_{cmbn}}$ las extensiones de las funciones $\mathtt{f}_{invar_{gen}}$ y $\mathtt{f}_{igual_{gen}}$, respectivamente. Es decir, $\mathtt{f}_{invar_{cmbn}}\colon \mathtt{T}_{cmbn} \to \mathcal{U}$ es la función total definida por $\mathtt{f}_{invar_{cmbn}}\big(\langle \mathtt{p}, [(\mathtt{t}_1,\mathtt{x}_1),\ldots,(\mathtt{t}_\mathtt{m},\mathtt{x}_\mathtt{m})]\rangle\big) \neq_{\mathtt{eq}} \mathtt{nil}$ si y solo si se verifican:

i) $\mathtt{f}_{invar_{gen}}(\mathtt{p},\mathtt{x}_\mathtt{i}) \neq_{\mathtt{eq}} \mathtt{nil}$, para todo $\mathtt{i}\in\{\mathtt{1},\ldots,\mathtt{m}\}$

ii) $\mathtt{f}_{igual_{gen}}(\mathtt{p},\mathtt{x}_\mathtt{i},\mathtt{x}_\mathtt{j}) =_{\mathtt{eq}} \mathtt{nil}$, para todo $\mathtt{i},\mathtt{j}\in\{\mathtt{1},\ldots,\mathtt{m}\}$ con $\mathtt{i}\neq\mathtt{j}$

La función $\mathtt{f}_{igual_{cmbn}}\colon \mathtt{T}_{cmbn}\times\mathtt{T}_{cmbn} \to \mathcal{U}$ es parcial con dominio $\big\{(\mathtt{c1},\mathtt{c2}) \in \mathtt{T}_{cmbn}\times\mathtt{T}_{cmbn} \mid \mathtt{f}_{invar_{cmbn}}(\mathtt{c1}) \neq_{\mathtt{eq}} \mathtt{nil} \text{ y } \mathtt{f}_{invar_{cmbn}}(\mathtt{c2}) \neq_{\mathtt{eq}} \mathtt{nil}\big\}$ y se define como $\mathtt{f}_{igual_{cmbn}}\big(\langle \mathtt{p}, [(\mathtt{t}_1,\mathtt{x}_1),\ldots,(\mathtt{t}_\mathtt{m},\mathtt{x}_\mathtt{m})]\rangle, \langle \mathtt{q}, [(\mathtt{s}_1,\mathtt{y}_1),\ldots,(\mathtt{s}_\mathtt{n},\mathtt{y}_\mathtt{n})]\rangle\big) \neq_{\mathtt{eq}} \mathtt{nil}$ si y solo si se verifican:

i) `(= p q)`

ii) `(= n m)`

iii) para cada $\mathtt{i} \in \{1, \ldots, \mathtt{m}\}$, $\exists | \mathtt{j} \in \{1, \ldots, \mathtt{m}\}$ tal que

- `(= t`$_\mathtt{i}$ `s`$_\mathtt{j}$`)`
- $\mathtt{f}_{igual_{gen}}(\mathtt{p}, \mathtt{x}_\mathtt{i}, \mathtt{y}_\mathtt{j}) \neq_{\mathtt{eq}} \mathtt{nil}$

Por definición de la categoría $Imp_{ef}^{\mathtt{T}^{nat}}(\mathcal{T}_{\mathtt{CC}})$ la función $\mathtt{f}_{card_{gen}}$: `integer` $\rightarrow$ (`satisfies naturalp`) es total, y la función $\mathtt{f}_{enum_{gen}}$: `integer` $\times$ (`satisfies naturalp`) $\rightarrow \mathcal{U}$ es parcial con dominio $\{(\mathtt{p}, \mathtt{n}) \mid \mathtt{n} < \mathtt{f}_{card_{gen}}(\mathtt{p})\}$ y verificando:

i) para cada $(\mathtt{p}, \mathtt{n}) \in Def(\mathtt{f}_{enum_{gen}})$, se tiene que $\mathtt{f}_{invar_{gen}}(\mathtt{p}, \mathtt{f}_{enum_{gen}}(\mathtt{p}, \mathtt{n})) \neq_{\mathtt{eq}} \mathtt{nil}$

ii) para cada par $(\mathtt{p}, \mathtt{n1}), (\mathtt{p}, \mathtt{n2}) \in Def(\mathtt{f}_{enum_{gen}})$, se verifica

$$\mathtt{f}_{igual_{gen}}(\mathtt{p}, \mathtt{f}_{enum_{gen}}(\mathtt{p}, \mathtt{n1}), \mathtt{f}_{enum_{gen}}(\mathtt{p}, \mathtt{n2})) =_{\mathtt{eq}} \mathtt{nil}$$

iii) para cada $(\mathtt{p}, \mathtt{g}) \in$ `integer` $\times \mathcal{U}$ tal que $\mathtt{f}_{invar_{gen}}(\mathtt{p}, \mathtt{g}) \neq_{\mathtt{eq}} \mathtt{nil}$, existe $\mathtt{n} \in$ `(satisfies naturalp)` con $(\mathtt{p}, \mathtt{n}) \in Def(\mathtt{f}_{enum_{gen}})$ y tal que $\mathtt{f}_{igual_{gen}}(\mathtt{p}, \mathtt{g}, \mathtt{f}_{enum_{gen}}(\mathtt{p}, \mathtt{n})) \neq_{\mathtt{eq}} \mathtt{nil}$

Las operaciones de $\mathcal{M}^{ef,\underline{\mathtt{T}},red,gen}$ que provienen de las operaciones de la signatura `CC`, de los invariantes o de las igualdades de los generadores, se definen como sus correspondientes en $\mathcal{M}^{rep,\underline{\mathtt{T}},red,gen}$ (dadas en la página 235). Respecto a las enumeraciones de representantes canónicos de los generadores éstas vienen definidas por las proyecciones en las componentes correspondientes, que, explícitamente vienen dadas como sigue:

- $rep\_card_{gen_{\mathcal{M}^{ef,\underline{\mathtt{T}},red,gen}}}$: $\mathcal{M}_{imp_{\mathtt{CC}}}^{ef,\underline{\mathtt{T}},red,gen} \times$ `integer` $\rightarrow$ (`satisfies naturalp`) es la función total dada por $rep\_card_{gen_{\mathcal{M}^{ef,\underline{\mathtt{T}},red,gen}}}(\overline{\mathtt{f}}, \mathtt{p}) := \mathtt{f}_{card_{gen}}(\mathtt{p})$.
- La función $rep\_enum_{gen_{\mathcal{M}^{ef,\underline{\mathtt{T}},red,gen}}}$: $\mathcal{M}_{imp_{\mathtt{CC}}}^{ef,\underline{\mathtt{T}},red,gen} \times$ `integer` $\times$ (`satisfies naturalp`) $\rightarrow \mathcal{U}$ es parcial con dominio el conjunto $\{(\overline{\mathtt{f}}, \mathtt{p}, \mathtt{n}) \mid \mathtt{n} < \mathtt{f}_{card_{gen}}(\mathtt{p})\}$ y se define como $rep\_enum_{gen_{\mathcal{M}^{ef,\underline{\mathtt{T}},red,gen}}}(\overline{\mathtt{f}}, \mathtt{p}, \mathtt{n}) := \mathtt{f}_{enum_{gen}}(\mathtt{p}, \mathtt{n})$.

De nuevo, podemos dar una implementación canónica $\mathcal{I}^{ef,\underline{\mathtt{T}},red,gen,can}$ de $\mathcal{M}^{ef,\underline{\mathtt{T}},red,gen}$, que resultará ser objeto final en la categoría de familias de álgebras definidas a partir de implementaciones de $Imp_{ef}^{\mathtt{T}^{nat}}(\mathcal{T}_{\mathtt{CC}})$. En este ejemplo viene dada como sigue:

- Representaciones literales sobre los tipos `integer`, $\mathtt{T}_{cmbn}$, `(satisfies naturalp)` y $\mathcal{U}$.
- Para dar la representación del género $imp_{\mathtt{CC}}$, nos apoyamos en el siguiente tipo

```
(defstruct REP-ef-IMP-CC rep-card-gen rep-enum-gen (:include
                                    REP-IMP-CC-red-gen))
```

y tomamos como dominio el formado por los objetos Lisp del tipo anterior con todos sus campos objetos funcionales, es decir, los objetos Lisp que satisfacen el siguiente predicado

```
(defun D-REP-ef-IMP-CC-p (x)
     (and (typep x 'REP-ef-IMP-CC)
          (functionp (REP-ef-IMP-CC-rep-invar-gen x))
          (functionp (REP-ef-IMP-CC-rep-igual-gen x))
          (functionp (REP-ef-IMP-CC-rep-card-gen x))
          (functionp (REP-ef-IMP-CC-rep-enum-gen x))
          (functionp (REP-ef-IMP-CC-imp-dffr x))))
```

El soporte $\mathcal{S}^{ef,\underline{\mathtt{T}},red,gen,can}_{imp_{\mathtt{CC}}}$ viene definido como sigue

$\mathcal{S}^{ef,\underline{\mathtt{T}},red,gen,can}_{imp_{\mathtt{CC}}} = \Big\{ \mathtt{x} \in$ `(satisfies D-REP-ef-IMP-CC)` $\mid \exists\ \mathcal{I} = (\underline{\mathcal{R}}, (\mathtt{p}^{\mathcal{I}}_{\sigma})_{\sigma\in\{suma,opp,zero,mult,dffr\}}, (\mathtt{p}^{\mathcal{I}}_{invar_g})_{g\in\{int,cmbn\}}, (\mathtt{p}^{\mathcal{I}}_{igual_g})_{g\in\{int,cmbn\}}, \mathtt{p}^{\mathcal{I}}_{card_{gen}}, \mathtt{p}^{\mathcal{I}}_{enum_{gen}}) \in Obj(Imp^{\underline{\mathtt{T}}^{nat}}_{ef}(\mathcal{T}_{\mathtt{CC}}))$ tal que denotando

$\mathcal{S}^{\mathcal{I}}_{gen} = \big\{(\mathtt{p},\mathtt{g}) \in$ `integer` $\times\ \mathcal{U} \mid$ `(not (eq (funcall` $\mathtt{c}^{\mathcal{I}}_{invar_{cmbn}}$ `(make-cmb :dgr p :lst (list (make-mnm :cff 1 :gnr g)))) nil))`$\big\}$ y

$\mathcal{S}^{\mathcal{I}}_{cmbn} = \big\{\langle \mathtt{p}, [(\mathtt{t}_1,\mathtt{x}_1),\ldots,(\mathtt{t}_{\mathtt{m}},\mathtt{x}_{\mathtt{m}})]\rangle \in \mathtt{T}_{cmbn} \mid (\mathtt{p},\mathtt{x_i}) \in \mathcal{S}^{\mathcal{I}}_{gen}$ para todo $\mathtt{i} \in \{1,\ldots,\mathtt{m}\}$ y `(eq (funcall` $\mathtt{c}^{\mathcal{I}}_{igual_{cmbn}}$ `p` $\mathtt{x_i}$ $\mathtt{x_j}$`) nil)` para todo $\mathtt{i},\mathtt{j} \in \{1,\ldots,\mathtt{m}\}$ con $\mathtt{i} \neq \mathtt{j}\big\}$, se tiene que

- Existe un programa $\mathtt{p}^{\mathtt{x}}_{invar_{gen}}$ con soporte $\mathcal{S}^{\mathcal{I}}_{gen}$ verificando:
  - *i)* `(eq` $\mathtt{c}^{\mathtt{p}^{\mathtt{x}}_{invar_{gen}}}$ `(REP-ef-IMP-CC-rep-invar-gen x))`
  - *ii)* La función definida por el programa $\mathtt{p}^{\mathtt{x}}_{invar_{gen}}$ extendida por linealidad a $\mathtt{T}_{cmbn}$, coincide con la función definida por el programa $F(\mathtt{p}^{\mathcal{I}}_{invar_{cmbn}})$
- Existe un programa $\mathtt{p}^{\mathtt{x}}_{igual_{gen}}$ con soporte el conjunto $\big\{(\mathtt{p},\mathtt{g1},\mathtt{g2}) \in$ `integer` $\times\ \mathcal{U} \times \mathcal{U} \mid (\mathtt{p},\mathtt{g1}),(\mathtt{p},\mathtt{g2}) \in \mathcal{S}^{\mathcal{I}}_{gen}\big\}$ verificando:
  - *i)* `(eq` $\mathtt{c}^{\mathtt{p}^{\mathtt{x}}_{igual_{gen}}}$ `(REP-ef-IMP-CC-rep-igual-gen x))`
  - *ii)* La función definida por el programa $\mathtt{p}^{\mathtt{x}}_{igual_{gen}}$ extendida por linealidad al conjunto $\big\{(\mathtt{c1},\mathtt{c2}) \in \mathcal{S}^{\mathcal{I}}_{cmbn} \times \mathcal{S}^{\mathcal{I}}_{cmbn} \mid$ `(= (cmb-dgr c1) (cmb-dgr c2))`$\big\}$ coincide con la función definida por el programa $F(\mathtt{p}^{\mathcal{I}}_{igual_{cmbn}})$
- `(eq` $\mathtt{c}^{\mathtt{p}^{\mathcal{I}}_{card_{gen}}}$ `(REP-ef-IMP-CC-rep-card-gen x))`
- `(eq` $\mathtt{c}^{\mathtt{p}^{\mathcal{I}}_{enum_{gen}}}$ `(REP-ef-IMP-CC-rep-enum-gen x))`

· el $\Omega_{\mathtt{CC}}$-conjunto de $\mathcal{I}$ es $\big(\{(\mathtt{c1},\mathtt{c2}) \in \mathcal{S}^{\mathcal{I}}_{cmbn} \times \mathcal{S}^{\mathcal{I}}_{cmbn} \mid \mathtt{(= (cmb\text{-}dgr\ c1)\ (cmb\text{-}dgr\ c2))}\ \}, \mathcal{S}^{\mathcal{I}}_{cmbn}, \mathtt{integer}, \mathtt{integer} \times \mathcal{S}^{\mathcal{I}}_{cmbn}, \mathcal{S}^{\mathcal{I}}_{cmbn}\big)$

· $\mathtt{(eq\ \ c}^{\mathtt{p}^{\mathcal{I}}_{dffr}}\ \mathtt{(REP\text{-}ef\text{-}IMP\text{-}CC\text{-}imp\text{-}dffr\ \ x))}$ $\Big\}$

La función de abstracción $\alpha^{ef,\underline{\mathtt{T}},red,gen,can}_{imp_{\mathtt{CC}}} : \mathcal{S}^{ef,\underline{\mathtt{T}},red,gen,can}_{imp_{\mathtt{CC}}} \to \mathcal{M}^{ef,\underline{\mathtt{T}},red,gen}_{imp_{\mathtt{CC}}}$ lleva cada dato $\mathtt{x}$ del soporte a la tupla de funciones $\big(\mathtt{f}^{\mathtt{x}}_{invar_{gen}}, \mathtt{f}^{\mathtt{x}}_{igual_{gen}}, \mathtt{f}^{\mathtt{x}}_{card_{gen}}, \mathtt{f}^{\mathtt{x}}_{enum_{gen}}, \mathtt{f}^{\mathtt{x}}_{dffr}\big)$ definidas por los códigos almacenados en los campos de $\mathtt{x}$ sobre los dominios $\big(\mathcal{S}^{\mathcal{I}}_{gen}, \{(\mathtt{p},\mathtt{g1},\mathtt{g2}) \in \mathtt{integer} \times \mathcal{U} \times \mathcal{U} \mid (\mathtt{p},\mathtt{g1}),(\mathtt{p},\mathtt{g2}) \in \mathcal{S}^{\mathcal{I}}_{gen}\}, \mathtt{integer}, \{(\mathtt{p},\mathtt{n}) \mid \mathtt{n} < \mathtt{f}^{\mathtt{x}}_{card_{gen}}(\mathtt{p})\}, \mathcal{S}^{\mathcal{I}}_{cmbn}\big)$, respectivamente.

Tomamos la igualdad de la abstracción como igualdad de la representación.

- Como programas que implementan a las operaciones que también lo son del álgebra $\mathcal{M}^{rep,\underline{\mathtt{T}},red,gen}$, tomamos los mismos que los de la implementación $\mathcal{I}^{rep,\underline{\mathtt{T}},red,gen,can}$. Para las enumeraciones, consideramos los siguientes:

  · El programa $\mathtt{p}^{ef,\underline{\mathtt{T}},red,gen,can}_{rep\_card_{gen}}$ tiene por soporte $\mathcal{S}^{ef,\underline{\mathtt{T}},red,gen,can}_{imp_{\mathtt{CC}}} \times \mathtt{integer}$ y por código el siguiente objeto funcional

  $$\mathtt{c}^{\mathtt{p}^{ef,\underline{\mathtt{T}},red,gen,can}_{rep\_card_{gen}}} \equiv \mathtt{\#'(lambda\ (x\ p)\ (funcall\ (REP\text{-}ef\text{-}IMP\text{-}CC\text{-}rep\text{-}card\text{-}gen\ x)\ p))}$$

  · el programa $\mathtt{p}^{ef,\underline{\mathtt{T}},red,gen,can}_{rep\_enum_{gen}}$ tiene por soporte al conjunto

  $$\big\{(\mathtt{x},\mathtt{p},\mathtt{n}) \in \mathcal{S}^{ef,\underline{\mathtt{T}},red,gen,can}_{imp_{\mathtt{CC}}} \times \mathtt{integer} \times (\mathtt{satisfies\ naturalp}) \mid \mathtt{(< n\ (funcall\ (REP\text{-}ef\text{-}IMP\text{-}CC\text{-}rep\text{-}card\text{-}gen\ x)\ p))}\big\}$$

  conjunto bien definido puesto que el soporte del programa anterior es $\mathcal{S}^{ef,\underline{\mathtt{T}},red,gen,can}_{imp_{\mathtt{CC}}} \times \mathtt{integer}$; el código del programa es el siguiente objeto funcional

  $$\mathtt{c}^{\mathtt{p}^{ef,\underline{\mathtt{T}},red,gen,can}_{rep\_enum_{gen}}} \equiv \mathtt{\#'(lambda\ (x\ p\ n)\ (funcall\ (REP\text{-}ef\text{-}IMP\text{-}CC\text{-}rep\text{-}card\text{-}gen\ x)\ p\ n))}$$

Observar que en el caso efectivo, como es el que nos ocupa, para definir los dominios de los programas de la signatura de partida se usan los invariantes, igual que en el caso localmente efectivo, y no las enumeraciones. El fin de éstas es otro, no tiene que ver con lo anterior, sino con la posibilidad de obtener información global acerca de la estructura a la que está representando.

Como ya comentamos, EAT no recoge en las estructuras de datos a los invariantes por motivos de eficiencia, lo que nos devuelve, de nuevo, al capítulo anterior en el que es necesario fijar los dominios de los programas a priori, ya que en las estructuras de datos no hay información suficiente para definirlos. Ahora bien, como ya hemos comentado en el caso localmente efectivo, este comentario se aplica desde el punto de vista teórico ya que en la práctica, no se requieren explícitamente los dominios de los programas.

Sin embargo, para acercarnos más a EAT queda un comentario pendiente y es que las enumeraciones no se codifican en EAT por medio de las dos funciones *card* y *enum* con las que hemos trabajado. En EAT se codifican utilizando el tipo `list` de Common Lisp de forma más constructiva, tal y como a continuación explicamos.

Hemos dado una enumeración de un conjunto como un par de funciones $card\colon \rightarrow$ (`satisfies naturalp`) y $enum\colon$ (`satisfies naturalp`) $\rightarrow \mathcal{U}$. Podemos observar que tienen la misma aridad que las que forman la signatura puramente observacional propuesta para las listas finitas en la sección 4.3 y la misma intención. Esto nos lleva a identificar las dos funciones anteriores que recogen una enumeración de un conjunto, con una lista (de longitud $card()$ y en la que el elemento que ocupa la posición i es el dato $enum(i)$). Generalizando esto a enumeraciones de conjuntos graduados en $\mathcal{U}$, una forma de dar una enumeración de un conjunto graduado `G` es por medio de una función `cbs`: `integer` $\rightarrow$ `list` de forma que, para cada `p` $\in$ `integer`, la imagen `cbs(p)` es una lista Lisp conteniendo los elementos del conjunto de grado `p`.

Para finalizar, vamos a incorporar esta última simplificación al ejemplo de los complejos de cadenas. Recordar que las enumeraciones que se manejan en este caso son enumeraciones del conjunto de representantes canónicos de un PER graduado ($\mathtt{G}$, $=_{\mathtt{G}}$), que se supone define el conjunto de generadores en cada grado del complejo de cadenas.

Así, la enumeración del conjunto de representantes canónicos del cociente $\mathtt{G}/_{=_{\mathtt{G}}}$, el conjunto graduado $\mathtt{Can_G}$, que hemos implementado mediante el par de funciones ($\mathtt{card_G}$: `integer` $\rightarrow$ (`satisfies naturalp`), $\mathtt{enum_G}$: `integer` $\times$ (`satisfies naturalp`) $\rightarrow \mathcal{U}$), podemos darla de forma explícita. Así, definimos una función total $\mathtt{cbs_G}$: `integer` $\rightarrow$ `list` dada, para cada `p` $\in$ `integer`, por $\mathtt{cbs_G(p)}:= \mathtt{l_p}$ siendo $\mathtt{l_p}$ de tipo `list` tal que (`length` $\mathtt{l_p}$) es $\mathtt{card_G(p)}$ y siendo, para cada `i` $\in \{\mathtt{0}, \ldots, \mathtt{card_G(p)-1}\}$, el elemento que ocupa la posición `i` en la lista $\mathtt{l_p}$ (`(nth i` $\mathtt{l_p}$`)`), el dato $\mathtt{enum_G(p, i)}$.

Esto nos permite dar otra descripción de las implementaciones de la categoría $Imp_{ef}^{\mathtt{I}^{nat}}(\mathcal{T}_{\mathtt{CC}})$ como una estructura

$$\mathcal{I} = \Big(\underline{\mathcal{R}},\ (\mathtt{p}^{\mathcal{I}}_{\sigma})_{\sigma\in\{suma,opp,zero,mult,dffr\}},\ (\mathtt{p}^{\mathcal{I}}_{invar_{int}},\ \mathtt{p}^{\mathcal{I}}_{invar_{cmbn}}),\ (\mathtt{p}^{\mathcal{I}}_{igual_{int}},\ \mathtt{p}^{\mathcal{I}}_{igual_{cmbn}}),\ \mathtt{p}^{\mathcal{I}}_{cbs_{gen}}\Big)$$

que se diferencia de la descripción que teníamos hasta ahora en que ésta aporta una enumeración explícita, es decir, el programa $\mathtt{p}^{\mathcal{I}}_{cbs_{gen}}$ implementa al conjunto de representantes canónicos de los generadores del complejo de cadenas al que implementa $\mathcal{I}$, de forma que la función que define tiene aridad $F(\mathtt{p}^{\mathcal{I}}_{cbs_{gen}})$: `integer` $\rightarrow$ `list` y proporciona la lista de los generadores del grado correspondiente. Las propiedades que verifica por ser enumeración, pueden escribirse en este caso, del siguiente modo:

i) para cada `p` $\in$ `integer`, se tiene que

$$F(\mathtt{p}^{\mathcal{I}}_{invar_{cmbn}})\big(\langle \mathtt{p},\ [(\mathtt{1},\ \mathtt{(nth\ i}\ F(\mathtt{p}^{\mathcal{I}}_{cbs_{gen}}(\mathtt{p}))\mathtt{)})]\rangle\big) \neq_{\mathtt{eq}} \mathtt{nil}$$

para todo `i` $\in \{\mathtt{0}, \ldots, \mathtt{(-\ (length}\ F(\mathtt{p}^{\mathcal{I}}_{cbs_{gen}}(\mathtt{p}))\ \ \mathtt{1)}\}$

ii) para cada `p` $\in$ `integer`, se tiene que

$$F(\mathtt{p}^{\mathcal{I}}_{igual_{cmbn}})\big(\langle \mathtt{p},\ [(\mathtt{1},\ \mathtt{(nth\ \ i}\ F(\mathtt{p}^{\mathcal{I}}_{cbs_{gen}}(\mathtt{p}))\mathtt{)})]\rangle, \langle \mathtt{p},\ [(\mathtt{1},\ \mathtt{(nth\ \ j}\ F(\mathtt{p}^{\mathcal{I}}_{cbs_{gen}}(\mathtt{p}))\mathtt{)})]\rangle\big) =_{\mathtt{eq}} \mathtt{nil}$$

para todo `i`, `j` $\in \{\mathtt{0}, \ldots, \mathtt{(-\ (length}\ F(\mathtt{p}^{\mathcal{I}}_{cbs_{gen}}(\mathtt{p}))\ \ \mathtt{1)}\}$ con `i` $\neq$ `j`

iii) para cada $\mathtt{p} \in \mathtt{integer}$ y para cada $\mathtt{i} \in \{\mathtt{0}, \ldots, \mathtt{(- \ (length}\ F(\mathtt{p}^{\mathcal{I}}_{cbs_{gen}}(\mathtt{p}))) \quad \mathtt{1)}\}$, existe $\mathtt{n} \in \mathtt{(satisfies\ naturalp)}$ con $\mathtt{n} \in \{\mathtt{0}, \ldots, \mathtt{(- \ (length}\ F(\mathtt{p}^{\mathcal{I}}_{cbs_{gen}}(\mathtt{p}))) \quad \mathtt{1)}\}$ tal que

$$F(\mathtt{p}^{\mathcal{I}}_{igual_{cmbn}})(\langle \mathtt{p}, [(\mathtt{1}, \mathtt{(nth\ i}\ F(\mathtt{p}^{\mathcal{I}}_{cbs_{gen}}(\mathtt{p}))))]\rangle, \langle \mathtt{p}, [(\mathtt{1}, \mathtt{(nth\ n}\ F(\mathtt{p}^{\mathcal{I}}_{cbs_{gen}}(\mathtt{p}))))]\rangle) \neq_{\mathtt{eq}} \mathtt{nil}$$

Esto nos permite dar otra descripción de la $\mathtt{CC}_{imp_{ef}}$-álgebra $\mathcal{M}^{ef,\underline{\mathtt{T}}}$, descripción que denotamos por $\mathcal{M}^{ef,\underline{\mathtt{T}},EAT}$ con soporte para el género $imp_{\mathtt{CC}}$ el siguiente conjunto

$$\mathcal{M}^{ef,\underline{\mathtt{T}},EAT}_{imp_{\mathtt{CC}}} = \Big\{(\mathtt{f}_{invar_{gen}}, \ \mathtt{f}_{igual_{gen}}, \ \mathtt{f}_{cbs_{gen}} \colon \mathtt{integer} \ \rightarrow \ \mathtt{list}, \ \mathtt{f}_{dffr}) \quad | \quad \exists \ \mathcal{I} \ \in \ Obj(Imp^{\underline{\mathtt{T}}^{nat}}_{ef}(\mathcal{T}_{\mathtt{CC}})) \ \text{ verificando que } \mathtt{f}_{invar_{cmbn}} = F(\mathtt{p}^{\mathcal{I}}_{invar_{cmbn}}), \mathtt{f}_{igual_{cmbn}} = F(\mathtt{p}^{\mathcal{I}}_{igual_{cmbn}}), \mathtt{f}_{cbs_{gen}} = F(\mathtt{p}^{\mathcal{I}}_{cbs_{gen}}) \text{ y } \mathtt{f}_{dffr} = F(\mathtt{p}^{\mathcal{I}}_{dffr})\Big\}$$

siendo $\mathtt{f}_{invar_{cmbn}}$ y $\mathtt{f}_{igual_{cmbn}}$ las extensiones por linealidad de las funciones $\mathtt{f}_{invar_{gen}}$ y $\mathtt{f}_{igual_{gen}}$, respectivamente.

El isomorfismo entre las álgebras $\mathcal{M}^{ef,\underline{\mathtt{T}},red,gen}$ y $\mathcal{M}^{ef,\underline{\mathtt{T}},EAT}$ viene dado por la equivalencia entre las dos formas que hemos visto de describir una enumeración finita: desde el punto de vista del comportamiento o desde el punto de vista constructivo.

La implementación canónica $\mathcal{I}^{ef,\underline{\mathtt{T}},EAT}$ de $\mathcal{M}^{ef,\underline{\mathtt{T}},EAT}$, objeto final en la categoría de familias de álgebras definidas a partir de implementaciones de $Imp^{\underline{\mathtt{T}}^{nat}}_{ef}(\mathcal{T}_{\mathtt{CC}})$ difiere de la implementación $\mathcal{I}^{ef,\underline{\mathtt{T}},red,gen,can}$ en que se apoya en el tipo

```
(defstruct EAT-IMP-CC cbs-gen (:include REP-IMP-CC-red-gen))
```

para definir el dominio del género $imp_{\mathtt{CC}}$. Los elementos del soporte de esta implementación son tuplas de objetos funcionales `x` del dominio correspondiente verificando las mismas condiciones impuestas a los del soporte de la implementación $\mathcal{I}^{ef,\underline{\mathtt{T}},red,gen,can}$, excepto las correspondientes a los campos `rep-card-gen` y `rep-enum-gen` (que no existen en este caso) y que se sustituyen por la siguiente condición

```
(eq c^{p^x_{cbs_gen}} (EAT-IMP-CC-cbs-gen x))
```

definiendo el programa correspondiente a esta operación de modo análogo al resto, es decir, por aplicación del objeto funcional almacenado en el campo correspondiente.

La descripción de la implementación canónica de $\mathcal{M}^{ef,\underline{\mathtt{T}},EAT}$ es la más cercana a EAT.

4.5 Estructuras mixtas en EAT: objetos con homología efectiva

Los objetos que EAT manipula son estructuras de datos cuyos campos son objetos funcionales. Diremos que uno de estos objetos es *efectivo* si nos permite obtener información global acerca de la estructura a la que está representando. Por ejemplo, en el caso de los complejos de cadenas,

esto corresponde con la existencia de un campo funcional que, en cada grado, nos devuelve la lista de representantes de los generadores. En otro caso, es decir, si no hay forma de obtener información global a partir de él, hablaremos de *objeto localmente efectivo*.

EAT puede trabajar con objetos efectivos y localmente efectivos que codifiquen estructuras algebraicas para la misma signatura. Para ello, además de los campos en los que se almacenan las operaciones procedentes de las estructuras algebraicas y de los que almacenan igualdades, añade otro campo (el que hemos denominado `cbs-gen`) que, en el caso de codificación efectiva, implementa a una enumeración explícita (en el caso de estar con una estructura graduada es un objeto funcional que en cada grado devuelve una lista) y, en el caso de codificación localmente efectiva, lo que se almacena en el campo es el símbolo `:LOCALLY-EFFECTIVE`.

En particular, si trabajamos con un complejo de cadenas y disponemos de información en el campo `cbs-gen` (que para los complejos de cadenas se llama `cbs` en EAT, véase [104]), es decir, si se trata de un complejo de cadenas efectivo, existe un algoritmo bien conocido (una primera referencia es [120] y otra más reciente [77]) que permite calcular, para cada grado $n \in \mathbb{Z}$, el correspondiente *grupo de homología* $H_n(C) := Ker\ d_n / Im\ d_{n+1}$, donde $d_* = \{d_n\}_{n\in\mathbb{Z}}$ denota el operador diferencial de C (puesto que por definición de complejo de cadenas $d_{n+1} \circ d_n = 0$, se sigue que el núcleo de d_n, $Ker\ d_n$, contiene a la imagen de d_{n+1} y, por tanto, el cociente $H_n(C)$ está bien definido, para todo $n \in \mathbb{Z}$). Dicho algoritmo está basado en la diagonalización de las matrices de enteros que representan a los operadores diferenciales. La homología es uno de los invariantes fundamentales (incluso fundacionales) de la Topología Algebraica y del Álgebra Homológica.

Por el contrario, si el campo `cbs` contiene el símbolo `:LOCALLY-EFFECTIVE`, no hay ninguna posibilidad de calcular su homología a partir de esa información (recuérdense los resultados de no calculabilidad del capítulo 3: es indecidible saber si el complejo de cadenas es trivial o no en un grado dado), incluso en el caso en que, por motivos teóricos, se conozca que sus grupos de homología son de tipo finito y, por tanto, potencialmente calculables. Precisamente los libros de Topología Algebraica están llenos de complejos de cadenas que no son de tipo finito, pero cuya homología sí lo es. La idea de Sergeraert consistió en codificar de modo localmente efectivo tales complejos, pero manteniendo un vínculo explícito con un complejo de cadenas efectivo que porta la misma información homológica que el complejo localmente efectivo. Más concretamente, un *complejo de cadenas con homología efectiva* es una terna $(C, \mathcal{E}, HC)$, donde C es un complejo de cadenas (normalmente codificado de modo localmente efectivo; siempre obviamente en el caso de que no sea de tipo finito alguno de sus grados), $\mathcal{E}$ es una equivalencia de homotopía explícita entre C y HC (véanse los detalles en [104] o [103], por ejemplo), codificada de modo funcional, y HC es un complejo de cadenas efectivo. La homología de un complejo de cadenas con homología efectiva es calculable (aplicando el algoritmo de diagonalización de matrices a HC, aún en el caso de que C no sea de tipo finito. Así pues, la terna $(C, \mathcal{E}, HC)$ define una estructura mixta: parte localmente efectiva (en ocasiones C; siempre $\mathcal{E}$) y parte efectiva (HC). La primera nos permitirá manipular la estructura algorítmicamente (pese a su infinitud esencial); la segunda nos permitirá realizar cálculos finales (cuando el usuario así lo solicite).

En general, un *objeto con homología efectiva* [114] se define en EAT como una cuaterna $(O, C(O), \mathcal{E}, HC)$, donde $C(O)$ es un complejo de cadenas canónicamente asociado a O y $(C(O), \mathcal{E}, HC)$ es un complejo de cadenas con homología efectiva. El caso más típico (e importante) aparece cuando O es igual a un conjunto simplicial K (puesto que se trata de una representación combinatoria de un espacio topológico). En ese caso, $C(K)$ es el complejo de cadenas asociado al conjunto simplicial K (véase la definición en [78], página 5) que será efectivo o localmente efectivo según lo sea K. Habitualmente, se parte de un conjunto simplicial K

efectivo, por ejemplo la esfera S^3 codificada como en el ejemplo 3.7.5 de la página 168, que da lugar de forma trivial (con equivalencia de homotopía la identidad) a un conjunto simplicial con homología efectiva $(S^3, C(S^3), id, C(S^3))$. Se le aplica entonces alguna construcción que interese en Topología Algebraica, como es la construcción espacio de lazos Ω (véase la definición en [78] o [103]) para obtener un nuevo conjunto simplicial con homología efectiva $(\Omega S^3, C(\Omega S^3), \mathcal{E}, HC)$. En este último objeto la situación ha variado sustancialmente puesto que ΩS^3 siendo un espacio de dimensión infinita en cada grado debe, obligatoriamente, ser localmente efectivo, y, por tanto, $C(\Omega S^3)$ también debe serlo. El programa EAT habrá construido tanto la equivalencia de homotopía $\mathcal{E}$ como el complejo de cadenas HC y para ello habrá tenido que crear y gestionar en tiempo de ejecución múltiples ejemplares de estructuras algebraicas (véase sección 1.1 y, en particular, la figura 1.3 que muestra la estructura de un objeto con homología efectiva), tratamiento que en esta memoria ha sido modelado gracias a la construcción que hemos denominado $()_{imp}$. Ahora, EAT es capaz de calcular grupos de homología del espacio topológico ΩS^3 (codificado como el conjunto simplicial con homología efectiva $(\Omega S^3, C(\Omega S^3), \mathcal{E}, HC)$. Pero, lo que es más importante, es capaz de seguir manipulándolo como estructura de datos para realizar nuevas operaciones interesantes en Topología Algebraica. En particular, el *mismo* algoritmo, puede ser aplicado a la misma estructura para obtener $(\Omega(\Omega S^3), C(\Omega(\Omega S^3)), \overline{\mathcal{E}}, \overline{HC})$, un nuevo conjunto simplicial con homología efectiva (véase la figura 1.3 en la página 10) y continuar trabajando con él hasta obtener informaciones de tipo homológico que no estaban disponibles por otros métodos hasta la fecha (véanse más detalles en [95] o [105], por ejemplo).

Conclusiones

Este trabajo concluye justo en el momento en que se acaban de introducir los objetos más interesantes de EAT: sus estructuras mixtas, los objetos con homología efectiva. A partir de ellos debería comenzar la descripción y modelado de los innovadores algoritmos implementados en EAT para el Cálculo Simbólico en Topología Algebraica. Afirmamos que para realizar dicha tarea las herramientas presentadas en esta memoria son suficientes. Puede ser una tarea laboriosa y no exenta de dificultades, pero creemos que lo esencial ha sido resuelto en este trabajo. En concreto, hemos alcanzado los siguientes objetivos:

1) Hemos mostrado que las especificaciones algebraicas estándar (al estilo clásico de [40] o [72]) son suficientes para modelar sistemas complejos como EAT, sin necesidad de recurrir a especificaciones con infinitos géneros, operaciones o axiomas o a formalismos de orden superior (como los presentados por ejemplo en [5] o [19]). Los beneficios de mantenerse en la línea más conservadora son evidentes: toda la tecnología desarrollada para las especificaciones algebraicas estándar [6] es directamente aplicable a nuestro caso. Como contrapartida, nos hemos visto obligados a enfrentarnos a diferentes problemas de *parcialidad*, que aunque son fácilmente superables, complican considerablemente los detalles técnicos de las construcciones y las demostraciones.

2) Hemos presentado las construcciones necesarias para modelar las estructuras de datos de EAT en el marco de la Teoría de Categorías, relacionándolas con construcciones categoriales más generales.

3) Hemos señalado la relación entre nuestra aproximación y algunos formalismos para la especificación algebraica orientada a objetos, como son las especificaciones ocultas (véanse [46], [23], [45], [27], [76], [94] y [47]) y las coalgebraicas (véanse [91], [106], [60], [61], [62], [76] y [27]). Nuestro problema resulta ser un caso particular que, siendo relevante, parece haber pasado desapercibido en la literatura. Puesto que en él no aparecen los problemas más difíciles de la orientación a objeto (por trabajar en un contexto de programación funcional pura y sin herencia), permite establecer una comparación muy precisa entre las diferentes propuestas, así como analizar con detalle las construcciones de objetos finales de la literatura, que en nuestro caso particular pueden expresarse de un modo sencillo, aclarando de paso el significado de las otras construcciones más complejas (como las que aparecen en [46] y [106]). En esta misma línea hemos explorado una relación entre la programación funcional, el enfoque coalgebraico y la implementación de estructuras de datos infinitas (presentes de modo esencial en EAT).

4) Basándonos en la clásica noción de representación de Hoare [58], hemos conseguido una formalización matemática completa de la noción de *implementación de un tipo abstracto*

de datos. Esta formalización nos ha permitido organizar las representaciones y las implementaciones en categorías, lo que ha facilitado sobremanera la expresión de los resultados posteriores. Como consecuencia, hemos mostrado que nuestra aproximación a la noción de implementación (que, como queda dicho, es heredada de la de Hoare) permite, por una parte, un tratamiento matemático consistente y, por otra, razonar sobre lenguajes de programación concretos (Common Lisp en nuestro caso). Queda así probada la superioridad para nuestro problema concreto, de este punto de vista frente a otras nociones más algebraicas de implementación (para las que el trabajo pionero es [39]; para un survey ver [85]) que, por mantener la coherencia con el formalismo de las especificaciones algebraicas (lo que les asegura la consistencia matemática), pierden el vínculo directo con los lenguajes de programación.

5) Hemos probado formalmente resultados de indecidibilidad y no calculabilidad que demuestran el hecho afirmado frecuentemente de modo informal (véase, por ejemplo, [98] o [97]) de que las representaciones localmente efectivas pierden información respecto a las efectivas y de que, en particular, en ellas no es posible implementar algunos operadores de naturaleza "global".

6) Hemos formalizado matemáticamente el carácter "universal" de las estructuras de datos de EAT, mostrando que forman parte de objetos finales en adecuadas categorías de implementaciones de tipos abstractos de datos. Informalmente, esto significa que dado cualquier otro modo de implementar siempre existirá una (única) forma de trasladarlo al "estilo EAT". Dada la importancia de este resultado, tanto matemáticamente como en la gestación de este trabajo (ver más adelante), consideramos el teorema 3.7.9 (y su consecuencia el teorema 3.7.11) como el central de la memoria. En su demostración confluyen todas las definiciones y técnicas introducidas previamente y también se utilizan las propiedades esenciales de la semántica de Common Lisp como lenguaje de programación funcional (en concreto, una forma particular de la β-reducción heredada del λ-cálculo).

7) Hemos establecido, en el marco de la especificación algebraica y de las implementaciones de tipos abstractos de datos, las diferencias esenciales entre las representaciones localmente efectivas (que son las que, por defecto, han aparecido a lo largo de la memoria) y las efectivas (que han sido tratadas en el último capítulo), formalizando de este modo los conceptos de *objeto localmente efectivo* y *objeto efectivo* que fueron introducidos por Sergeraert en [101] (véanse también [96] y [97]). Asímismo, hemos descrito brevemente los *objetos mixtos* de EAT (los objetos con homología efectiva) que tienen parte localmente efectiva y parte efectiva, objetos que, sin duda, proporcionan a EAT su aspecto más original y su mayor potencia de cálculo (respecto a sistemas que trabajasen en un contexto únicamente efectivo).

8) Por último, hemos mostrado cómo los desarrollos teóricos generales aportados pueden ser adaptados para tratar las estructuras concretas de EAT. Para ello nos hemos concentrado en dos de las estructuras fundamentales en EAT (y, de hecho, en cualquier sistema de Cálculo Simbólico en Topología Algebraica): los conjuntos simpliciales y los complejos de cadenas.

En resumen, consideramos alcanzado el objetivo general que nos habíamos propuesto: la especificación formal de un fragmento significativo de EAT (el que se refiere a las estructuras de datos "aisladas"), analizando sus propiedades estructurales y poniéndola en relación con múltiples trabajos que aparecen en la literatura.

En cuanto a la metodología utilizada se ha procurado emplear técnicas tan directas y sencillas como ha sido posible. En concreto, en lo relativo a la Teoría de Categorías hemos utilizado solo conceptos elementales (no usando, por ejemplo, las nociones de categorías fibradas o indexadas, que están muy relacionadas con nuestro enfoque), imprescindibles para expresar con concisión nuestros resultados. Este propósito de sencillez en la exposición no siempre ha sido fácil de conseguir, siendo un escollo de cierta importancia la necesidad de utilizar una notación compacta y no ambigua, en la que es necesario señalar explícitamente, en forma de superíndices y subíndices, muchos elementos relevantes (tipos de datos, dominios de definición, géneros, etc.). Por otra parte, pese a que las ideas intuitivas no son complejas, es necesario tener presentes constantemente varios niveles (el conceptual o de los modelos y funciones, el lógico o de los dominios y algoritmos y el físico o de los tipos concretos y códigos). Además, en cada uno de esos niveles hay que considerar una o varias igualdades (de los dominios, literales, naturales, de la representación, de la abstracción, etc.), que en ocasiones son coincidentes, pero en otras no. Para intentar controlar esta complejidad, y siempre con el objetivo de hacer comprensibles nuestros resultados, hemos preferido hacer una presentación "acumulativa": en el capítulo 2 hemos tratado el caso algebraico, sin mención ninguna a los lenguajes de programación; en el capítulo 3 estudiamos el caso de las implementaciones "puras" (es decir, solo se almacenan los códigos de los operadores, sin ninguna información explícitamente almacenada sobre los soportes de las representaciones); la aproximación del capítulo 3 es insuficiente para explicar la diferencia entre lo "localmente efectivo" y lo "efectivo", por lo que en el capítulo 4 incluimos entre lo almacenado en memoria los invariantes (o funciones de test), las igualdades de las representaciones y, en el caso efectivo, las enumeraciones de elementos. Esta presentación "por capas sucesivas" puede tener, pese a su objetivo de mantener la sencillez en la exposición, dos efectos indeseados. Por una parte, puede dejar en el lector una sensación de *déjà vu*, pues definiciones, resultados (como los de la existencia de coproductos y objetos finales) y ejemplos parecen repetirse en tres capítulos distintos. No obstante, debe tenerse en cuenta que cada capítulo desarrolla un aspecto diferente del problema (mejor dicho: cada capítulo se apoya en el anterior) y que hacer una redacción menos prolija hubiese significado exigir al lector una excesiva concentración en reconocer (y trasladar al nuevo contexto) ideas precedentes o bien limitarnos a hacer una exposición informal, sin el necesario rigor matemático. Por otra parte, la presentación lineal elegida puede hacer perder de vista las intuiciones y la gestación de nuestro trabajo. Por ello nos parece útil dar otra perspectiva de los capítulos, haciendo un poco de historia sobre nuestro tema de investigación.

Una primera versión del sistema EAT (que, de hecho, no llegó a ser plenamente operativa) contenía en cada estructura de datos un campo con la función de test (o invariante, según nuestra terminología, que es la de Hoare en [58]) que permitía decodificar el conjunto sobre el que el resto de operaciones trabajaban. Era natural comenzar así, pues era el reflejo directo de la *codificación funcional*, introducida por Sergeraert en [113]. Por otra parte, la función de test parecía la única justificación matemática de que se podía trabajar en el ordenador con conjuntos infinitos. Así pues si este punto de vista se hubiese impuesto, la versión recogida en nuestro capítulo 4 hubiese sido la que realmente daría lugar a la modelización más directa del sistema. Sin embargo, Sergeraert observó rápidamente que, desde el punto de vista operacional, la función de test es inútil. Puede ser útil en la fase de desarrollo, para comprobar que los "tipos" de los datos manejados son los adecuados, pero, supuestos correctos los programas del sistema si también lo son los datos de entrada, es obvio que aplicar las sucesivas funciones de test (comprobar la invariancia de las propiedades de los datos) sería un derroche innecesario. Puesto que EAT es un entorno de cálculo (y no de especificación) se decidió que las funciones de test no aparecieran en los códigos de los programas (como puede verse en la documentación

[104]). Dado que el test de igualdad tampoco ayuda en la tarea de determinar los conjuntos sobre los que se está trabajando, decidimos adoptar la aproximación estudiada en el capítulo 3: quedarnos únicamente con las tuplas de códigos de los operadores (que también aparecen en el estilo de programación de EAT). En este contexto demostramos el teorema 3.7.9, que fue el primer resultado positivo de relevancia encontrado y, por tanto, punto primordial en este trabajo. Pese a que el resultado es satisfactorio (véase el punto (6) más arriba), también es excesivamente rígido en algún sentido y no recoge plenamente el modo de trabajar en EAT (lo que hace que los ejemplos concretos del apartado 3.8 y del 2.9, tengan un cierto aire "artificial"). Para entender ese desajuste entre modelo y realidad, hay que tener presente que, en ausencia de funciones de test, solo el usuario es capaz de determinar (en su mente) sobre qué conjuntos está trabajando. Pero ese material "mental" resulta de poca utilidad a la hora de demostrar teoremas. Por ello, nosotros, en el capítulo 3, resolvimos la cuestión por el expeditivo método de fijar de una vez los conjuntos sobre los que se trabaja (los $\underline{\mathtt{T}}$ de ese capítulo o los D del caso algebraico en el capítulo 2) así como los dominios de definición de cada uno de los operadores (los Ω-conjuntos $\underline{\mathtt{S}}$). Así, pagando el precio de ser ciertamente inflexibles en lo que a los dominios se refiere, conseguimos mostrar que las implementaciones que son codificadas en EAT forman parte de objetos finales en ciertas categorías. Dicho resultado fue publicado en [67]. Uno de los referees nos indicó que estábamos utilizando técnicas no demasiado modernas (especificaciones algebraicas como en [40] y la noción de representación de Hoare [58]) y que nuestro punto de vista debería ser comparado con temas relacionados con la *semántica comportamental*. A raíz de dicho consejo (que resultó ser certero) comenzamos a estudiar literatura más reciente sobre especificaciones algebraicas y por medio de la técnica de ir eliminando de nuestra aproximación todos los detalles relativos a los lenguajes de programación, obtuvimos la relación con las especificaciones ocultas y las coálgebras que ha sido detallada en el capítulo 2 (y en [68]). Puesto que, como decimos, los enfoques de los capítulos 2 y 3 no recogen plenamente el modo de trabajar en EAT y, por otra parte, tratan únicamente el caso localmente efectivo, en el capítulo 4 recuperamos las funciones de test y las igualdades, obteniendo plena identificación con EAT, como puede verse en el tratamiento de los complejos de cadenas de la sección 4.4.5. (Pero, insistamos una vez más, si eliminamos las funciones de test, como sucede en EAT, no podemos obtener resultados teóricos y nos veríamos abocados a volver a la solución inicial en el tiempo: la propuesta en el capítulo 3.)

En lo que se refiere a líneas de continuación de este trabajo, existen múltiples posibilidades, algunas de las cuales han comenzado a ser tratadas en nuestro equipo de investigación. Algunas de estas vías ya han sido anunciadas a lo largo de la memoria. Una de ellas es generalizar los resultados aquí recogidos haciéndolos independientes del lenguaje de programación Common Lisp. Esta tarea no parece muy difícil mientras nos mantengamos dentro del paradigma de programación funcional. Algunos primeros tanteos en el paradigma orientado a objetos (concretamente en Java y C++) han sido publicados en [4].

Otro problema abierto, ya citado en la memoria, es abarcar fragmentos cada vez mayores de EAT. Concretamente, en este texto nos hemos centrado en estructuras de datos "aisladas" (como los complejos de cadenas, por ejemplo) sin ocuparnos de las relaciones entre distintas estructuras (técnicamente hablando, se trataría de estudiar signaturas ocultas con constructores desde datos o desde datos y objetos, y no únicamente deconstructoras como en esta memoria). Obviamente, sin incluir estas relaciones el sistema EAT no podrá ser completamente modelado. Unos primeros resultados, relativos al bloque conjuntos-simpliciales/complejos-de-cadenas, ya han sido publicados en [32]. Además, en dicho trabajo (así como en [33] y [34]) presentamos algunos de los resultados de esta memoria utilizando la noción de *institución* (ver, por ejemplo, [48], [30] o [117]). Todas las técnicas desarrolladas en esta memoria serán también de utilidad

a la hora de abordar el análisis del sistema Kenzo [36], sucesor de EAT. Pese a que nuestros resultados se aplican, con ligeras modificaciones, a Kenzo, serán insuficientes por sí solos para tratar todo Kenzo, pues está programado utilizando mucha de la riqueza de CLOS (Common Lisp Object System). En particular, la herencia y otros conceptos orientados a objetos juegan un papel esencial en Kenzo. (Algunos resultados preliminares sobre estos temas han sido publicados en [35].)

Para terminar, citar que otra línea de continuación de este trabajo es la aplicación de nuestras técnicas a otros sistemas de Cálculo Simbólico más generales, como Axiom [63] o Aldor [1].

Bibliografía

[1] *The Aldor programming language*. `http://www.aldor.org`.

[2] P. Aczel y N. Mendler. A Final Coalgebra Theorem. En D.H. Pitt, D.E. Rydeheard, P. Dybjer, A.M. Pitts y A. Poigne, editores, *Category theory and computer science,* Lecture Notes in Computer Science, vol. 389, pp. 357–365. Springer, 1989.

[3] V.S. Alagar y K. Periyasamy. *Specification of Software Systems*. Springer, 1998.

[4] J. Aransay, J.J. Olarte y J. Rubio. Implementación orientada a objetos de estructuras algebraicas: estudio de un caso en Java y C++. En J. Rubio, editor, *Actas del Séptimo Encuentro de Álgebra Computacional y Aplicaciones*, pp. 78–82. Universidad de La Rioja, 2001.

[5] E. Astesiano y M. Cerioli. Partial Higher-Order Specifications. *Fundamenta Informaticae*, 16(2):101–126, 1992.

[6] E. Astesiano, H.-J. Kreowski y B. Krieg-Brückner, editores. *Algebraic Foundations of Systems Specification*. Springer, 1999.

[7] H.P. Barendregt. Functional Programming and Lambda Calculus. En J.V. Leeuwen, editor, *Formal models and semantics*, pp. 321–364. Elsevier, 1992.

[8] M. Barr. Terminal coalgebras in well-founded set theory. *Theoretical Computer Science*, 114(2):299–315, 1993.

[9] M. Barr. Additions and corrections to "Terminal coalgebras in well-founded set theory". *Theoretical Computer Science*, 124(1):189–192, 1994.

[10] M. Barr y Ch. Wells. *Category Theory for Computing Science*. Prentice Hall International, 1995.

[11] Ch. Beierle y A. Voß. Implementation Specifications. En H.-J. Kreowski, editor, *Recent Trends in Data Type Specification*, vol. Informatik-Fachberichte 116, pp. 39–52. Springer, 1985.

[12] D. Bert, Ch. Choppy y P. Mosses, editores. *Recent Trends in Algebraic Development Techniques,* Lecture Notes in Computer Science, vol. 1827. Springer, 2000.

[13] H. Bidoit y R. Hennicker. Modular Correctness Proofs of Behavioural Implementations. *Acta Informatica*, 35(11):951–1005, 1998.

[14] R. Bird. *Introducción a la programación funcional con Haskell. – Segunda ed. –*. Prentice Hall, 2000.

[15] E. Bishop. *Foundations of constructive analysis*. McGraw-Hill, 1967.

[16] F. Borceux. *Handbook of Categorical Algebra 1. Basic Category Theory*. Cambridge University Press, 1994.

[17] F. Borceux. *Handbook of Categorical Algebra 2. Categories and Structures*. Cambridge University Press, 1994.

[18] E. Börger, editor. *Specification and Validation Methods*. Clarendon Press, 1995.

[19] M. Broy. Equational specification of partial higher-order algebras. *Theoretical Computer Science*, 57:3–45, 1988.

[20] M. Broy, B. Möller, P. Pepper y M. Wirsing. Algebraic implementations preserve program correctness. *Science of Computer Programming*, 7:35–53, 1986.

[21] M. Broy y M. Wirsing. Partial Abstract Types. *Acta Informatica*, (18):47–64, 1982.

[22] P. Burmeister. *A Model Theoretic Oriented Approach to Partial Algebras*. Akademie-Verlag, 1986.

[23] R. Burstall y R. Diaconescu. Hiding and Behaviour: an Institutional Approach. En A. William Roscoe, editor, *A Classical Mind: Essays in Honour of C.A.R. Hoare*, pp. 75–92. Prentice Hall, 1994.

[24] M. Cerioli, T. Mossakowski y H. Reichel. From Total Equational to Partial First-Order Logic. En E. Astesiano, H.-J. Kreowski y B. Krieg-Brückner, editores, *Algebraic Foundations of Systems Specification*, pp. 31–104. Springer, 1999.

[25] G.J. Chaitin. *The limits of Mathematics*. Springer, 1998.

[26] A. Church. *The calculi of lambda-conversion*. Princeton University Press, 1941.

[27] C. Cîrstea. Coalgebra Semantics for Hidden Algebra: Parameterised Objects and Inheritance. En F. Parisi-Presicce, editor, *Recent Trends in Algebraic Development Techniques,* Lecture Notes in Computer Science, vol. 1376, pp. 174–189. Springer, 1997.

[28] C. Cîrstea. Semantic Constructions for the Specification of Objects. En J. Fiadeiro, editor, *Recent Trends in Algebraic Development Techniques,* Lecture Notes in Computer Science, vol. 1589, pp. 63–78. Springer, 1999.

[29] W.R. Cook. Object-Oriented Programming versus Abstract Data Types. En J.W. Bakker, W.P. Roever y G. Rozenberg, editores, *Foundations of Object-Oriented Languages,* Lecture Notes in Computer Science, vol. 489, pp. 151–178. Springer, 1991.

[30] R. Diaconescu. Extra Theory Morphism for Institutions: Logical Semantics. *Applied Categorical Structures*, 6(4):427–453, 1998.

[31] R. Diaconescu y K. Futatsugi. Behavioural Coherence in Object-Oriented Algebraic Specification. *Journal of Universal Computer Science*, 6(1):74–96, 2000.

[32] C. Domínguez, L. Lambán, V. Pascual y J. Rubio. Hidden Specification of a Functional System. En R. Moreno-Díaz, B. Buchberger y J.-L. Freire, editores, *Computer Aided Systems Theory - EUROCAST 2001,* Lecture Notes in Computer Science, vol. 2178. Springer, 2001.

[33] C. Domínguez, L. Lambán, V. Pascual y J. Rubio. Instituciones: Matemáticas para la Especificación en Computación. En L. Español y J.L. Varona, editores, *Margarita Mathematica en memoria de José Javier (Chicho) Guadalupe Hernández*, pp. 221–233. Servicio de Publicaciones, Universidad de La Rioja, 2001.

[34] C. Domínguez, L. Lambán, V. Pascual y J. Rubio. Modelling Implementations as Institution Morphisms. En F. Orejas, F. Cuartero y D. Cazorla, editores, *PROLE 2001. Jornadas sobre Programación y Lenguajes*, pp. 229–239. Universidad de Castilla-La Mancha, 2001.

[35] C. Domínguez y J. Rubio. Modeling inheritance as coercion in a Symbolic Computation System. En B. Mourrain, editor, *Proceedings of the 2001 International Symposium on Symbolic and Algebraic Computation (ISSAC'2001)*, pp. 107–115. ACM Press, 2001.

[36] X. Dousson, F. Sergeraert y Y. Siret. *The Kenzo program*. Institut Fourier, Grenoble, 1999. `http://www-fourier.ujf.grenoble.fr/~sergerar/Kenzo/`.

[37] H. Ehrig, M. Große-Rhode y U. Wolter. On the Role of Category Theory in the Area of Algebraic Specifications. En M. Haveraaen, O. Owe y O.-J. Dahl, editores, *Recent Trends in Data Type Specification,* Lecture Notes in Computer Science, vol. 1130, pp. 17–45. Springer, 1995.

[38] H. Ehrig, M. Große-Rhode y U. Wolter. Applications of Category Theory to the Area of Algebraic Specification in Computer Science. *Applied Categorical Structures*, 6(1):1–35, 1998.

[39] H. Ehrig, H.J. Kreowski, B. Mahr y P. Padawitz. Algebraic implementation of abstract data types. *Theoretical Computer Science*, 20:209–263, 1982.

[40] H. Ehrig y B. Mahr. *Fundamentals of Algebraic Specifications 1. Equations and Initial Semantics*. Springer, 1985.

[41] S. Eilenberg y S. Mac Lane. General theory of natural equivalences. *Transactions of the American Mathematical Society*, 58:231–294, 1945.

[42] J.L. Fiadeiro, editor. *Recent Trends in Algebraic Development Techniques,* Lecture Notes in Computer Science, vol. 1589. Springer, 1999.

[43] J. Goguen. Types as theories. En G.M. Reed, A.W. Roscoe y R.F. Wachter, editores, *Topology and Category Theory in Computer Science*, pp. 357–390. Oxford University Press, 1991.

[44] J. Goguen y R. Diaconescu. Towards an Algebraic Semantics for the Object Paradigm. En H. Ehrig y F. Orejas, editores, *Recent Trends in Data Type Specification,* Lecture Notes in Computer Science, vol. 785, pp. 1–29. Springer, 1994.

[45] J. Goguen y G. Malcolm. Hidden Coinduction: Behavioral Correctness Proofs for Objects. *Mathematical Structures in Computer Science*, 9(3):287–319, 1999.

[46] J. Goguen y G. Malcolm. A hidden agenda. *Theoretical Computer Science*, 245(1):55–101, 2000.

[47] J. Goguen y G. Roçu. Hiding More of Hidden Algebra. En J.M. Wing, J. Woodcook y J. Davies, editores, *FM99 - Formal Methods,* Lecture Notes in Computer Science, vol. 1709, pp. 1704–1719. Springer, 1999.

[48] J.A. Goguen y R.M. Burstall. Institutions: Abstract Model Theory for Specification and Programming. *Journal of the Association for Computing Machinery*, 39(1):95–146, 1992.

[49] J.A. Goguen, J.W. Thatcher y E.G. Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. *Current Trends in Programming Methodology, IV: Data Structuring*, pp. 80–144, 1978.

[50] J.A. Goguen, J.W. Thatcher, E.G. Wagner y J.B. Wright. Abstract data types as initial algebras and the correctness of data representations. En *Proceedings of Conference of Computer Graphics, Pattern Recognition and Data Structures*, pp. 89–93. ACM, 1975.

[51] J.A. Goguen, J.W. Thatcher, E.G. Wagner y J.B. Wright. Initial algebra semantics and continuous algebras. *Journal of the Association for Computing Machinery*, (24):68–95, 1977.

[52] J.V. Guttag. Abstract data types and the development of data structures. En *Suplement to Proceedings of the Conference on Data Abstraction, Definition and Structure*, vol. 8. SIGPLAN notices, 1976.

[53] J.V. Guttag. Abstract data types and the development of data structures. *Communications of the ACM*, 20:396–404, 1978.

[54] J.V. Guttag y J.J. Horning. The Algebraic Specification of Abstract Data Types. *Acta Informatica*, (10):27–52, 1978.

[55] M. Haveraaen y E.G. Wagner. Guarded Algebras: Disguising Partiality so you won't know whether its there. En D. Bert, Ch. Choppy y P. Mosses, editores, *Recent Trends in Algebraic Development Techniques,* Lecture Notes in Computer Science, vol. 1827, pp. 182–200. Springer, 1999.

[56] R. Hennicker. Implementation of Parameterized Observational Specifications. En J. Díaz y F. Orejas, editores, *TAPSOFT'89. Proceedings of the International Joint Conference on Theory and Practice of Software Development,* Lecture Notes in Computer Science, vol. 351, pp. 290–305. Springer, 1989.

[57] R. Hennicker y Ch. Schmitz. Object-Oriented Implementation of Abstract Data Type Specifications. En M. Wirsing y M. Nivat, editores, *Algebraic Methodology and Software Technology (AMAST96),* Lecture Notes in Computer Science, vol. 1101, pp. 163–179. Springer, 1996.

[58] C.A.R. Hoare. Proofs of Correctness of Data Representations. *Acta Informatica*, 1:271–281, 1972.

[59] J.E. Hopcroft y J.D. Ullman. *Introduction to automata theory, languages y computation.* Addison-Wesley, 1979.

[60] B. Jacobs. Mongruences and Cofree Coalgebras. En V.S. Alagar y M. Nivat, editores, *Algebraic Methodology and Software Technology (AMAST95),* Lecture Notes in Computer Science, vol. 936, pp. 245–260. Springer, 1995.

[61] B. Jacobs. Objects and Classes, Co-algebraically. En B. Freitag, C.B. Jones, C. Lengauer y H.-J. Schek, editores, *Object-Orientation with Parallelism and Persistence*, pp. 83–103. Kluwer Academic, 1996.

[62] B. Jacobs y J. Rutten. A Tutorial on (Co)algebras and (Co)induction. *EATCS Bulletin*, 62:222–259, 1997.

[63] R.D. Jenks. *AXIOM: the scientific computation system.* Springer, 1992.

[64] L. Lambán, V. Pascual y J. Rubio. Categorías de implementaciones de Tipos Abstractos de Datos. En *Actas del Segundo Encuentro de Álgebra Computacional y Aplicaciones*, pp. 86–94, Universidad de Sevilla, 1996.

[65] L. Lambán, V. Pascual y J. Rubio. Resultados de no calculabilidad en representaciones de Tipos Abstractos de Datos. En *Actas del Tercer Encuentro de Álgebra Computacional y Aplicaciones*, pp. 133–140, Universidad de Granada, 1997.

[66] L. Lambán, V. Pascual y J. Rubio. Simplicial sets in the EAT system. En *Actas del Quinto Encuentro de Álgebra Computacional y Aplicaciones*, pp. 167–176, Universidad de La Laguna, 1999.

[67] L. Lambán, V. Pascual y J. Rubio. Specifying Implementations. En S. Dooley, editor, *Proceedings of the 1999 International Symposium on Symbolic and Algebraic Computation (ISSAC'99)*, pp. 245–251. ACM Press, 1999.

[68] L. Lambán, V. Pascual y J. Rubio. *An object-oriented interpretation of the EAT system.* Prepublicación, 2000.

[69] L. Lambán y J. Rubio. Tipos Abstractos de Datos en Álgebra Homológica. En L. González-Vega, editor, *Actas del Primer Encuentro de Álgebra Computacional y Aplicaciones*, pp. 113–122, Universidad de Cantabria, 1995.

[70] H.R. Lewis y C.H. Papadimitriou. *Elements of the Theory of Computation.* Prentice-Hall, 1981.

[71] J.D. Lipson. *Elements of Algebra and Algebraic Computing.* Benjamin/Cummings Publishing Company, Inc., 1981.

[72] J. Loeckx, H.D. Ehrich y M. Wolf. *Specification on Abstract Data Types.* Wiley–Teubner, 1996.

[73] S. Mac Lane. *Homology.– 4th ed. –.* Springer, 1994.

[74] S. Mac Lane. *Categories for the Working Mathematician.– 2nd ed. –.* Springer, 1998.

[75] T.S.E. Maibaum y C.J. Lucena. Higher-order data types. *International Journal of Computer and Information Sciences*, (9):31–53, 1980.

[76] G. Malcolm. Behavioural Equivalence, Bisimulation and Minimal Realisation. En O. Owe M. Haveraaen y O.J. Dahl, editores, *Recent Trends in Data Type Specification,* Lecture Notes in Computer Science, vol. 1130, pp. 359–378. Springer, 1996.

[77] C.R.F. Maunder. *Algebraic Topology.* Dover, 1996.

[78] J.P. May. *Simplicial Objects in Algebraic Topology.* Midway, 1982.

[79] C. McLarty. *Elementary Categories, Elementary Toposes.* Oxford University Press, 1995.

[80] R. Milner. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17(3):348–375, 1978.

[81] B. Möller. On the algebraic specification of infinite objects- Ordered and continuous models of algebraic types. *Acta Informatica*, (22):537–578, 1985.

[82] B. Möller. Algebraic Specifications with higher-order operators. En L. Meertens, editor, *Proceedings IFIP TC2 Working Conference on Program Specification and Transformation*, pp. 367–392, 1987.

[83] B. Möller, A. Tarlecki y M. Wirsing. Algebraic Specifications of Reachable Higher-Order Algebras. En D. Sannella y A. Tarlecki, editores, *Recent Trends in Data Type Specification,* Lecture Notes in Computer Science, vol. 332, pp. 154–169. Springer, 1988.

[84] N. Nissanke. *Formal Specification*. Springer, 1999.

[85] F. Orejas, M. Navarro y A. Sánchez. Implementation and Behavioural Equivalence: a Survey. En M. Bidoit y C. Choppy, editores, *Recent Trends in Data Type Specification,* Lecture Notes in Computer Science, vol. 655, pp. 93–125. Springer, 1993.

[86] F. Parisi-Presicce, editor. *Recent Trends in Algebraic Development Techniques,* Lecture Notes in Computer Science, vol. 1376. Springer, 1997.

[87] Ch.C. Pinter. *Set Theory*. Addison-Wesley, 1971.

[88] K. Pitman. *Common Lisp HyperSpec*. `http://www.harlequin.com/education/books/HyperSpec/`.

[89] H. Reichel. Behavioural equivalence- a unifying concept for initial and final specification methods. En M. Arató y L. Varga, editores, *Third Hungarian Computer Science Conference*, pp. 27–39, Budapest, 1981. Akademiai Kiado.

[90] H. Reichel. *Initial Computability, Algebraic Specifications and Partial Algebras*. Clarendon Press, 1987.

[91] H. Reichel. An approach to object semantics based on terminal coalgebras. *Mathematical Structures in Computer Science*, 5:129–152, 1995.

[92] H. Reichel. Specification semantics. En E. Astesiano, H.-J. Kreowski y B. Krieg-Brückner, editores, *Algebraic Foundations of Systems Specification*, pp. 131–158. Springer, 1999.

[93] J.C. Reynolds. *Theories of Programming Languages*. Cambridge University Press, 1998.

[94] G. Roçu y J. Goguen. Hidden Congruent Deduction. En R. Caferra y G. Salzer, editores, *Automated Deduction in Classical and Non-Classical Logics,* Lecture Notes in Artificial Intelligence, vol. 1761, pp. 251–266. Springer, 2000.

[95] J. Rubio. *Homologie Effective des espaces de lacets itérés: un logiciel*. Thèse, Institut Fourier, Grenoble, 1991.

[96] J. Rubio. Locally effective objects and Artificial Intelligence. *Revista Academia de Ciencias, Zaragoza*, 55:67–77, 2000.

[97] J. Rubio. Locally Effective Objects and Artificial Intelligence. En J.A. Campbell y E. Roanes-Lozano, editores, *Artificial Intelligence and Symbolic Computation,* Lecture Notes in Artificial Intelligence, vol. 1930, pp. 223–236. Springer, 2001.

[98] J. Rubio y F. Sergeraert. Constructive Algebraic Topology. *Aparecerá en Bulletin des Sciences Mathématiques.*

[99] J. Rubio y F. Sergeraert. Suites spectrales d'Eilenberg-Moore et homologie effective. *Comptes-Rendus l'Académie des Sciences*, pp. 723–726, 1988.

[100] J. Rubio y F. Sergeraert. Supports Acycliques et Algorithmique. *Astérisque*, 192:35–55, 1990.

[101] J. Rubio y F. Sergeraert. Locally effective objects and algebraic topology. En *Computational Algebraic Geometry*, pp. 235–251. Birkhäuser, 1993.

[102] J. Rubio y F. Sergeraert. *Homologie effective et problème d'Adams.* Institut Fourier, Grenoble, 1994. `http://www-fourier.ujf.grenoble.fr/~sergerar`.

[103] J. Rubio y F. Sergeraert. Constructive Algebraic Topology. *Lecture Notes Summer School on Fundamental Algebraic Topology*, Institut Fourier, Grenoble 1997.

[104] J. Rubio, F. Sergeraert y Y. Siret. *EAT: Symbolic Software for Effective Homology Computation.* Institut Fourier, Grenoble, 1997. `ftp://fourier.ujf-grenoble.fr/pub/EAT`.

[105] J. Rubio, F. Sergeraert y Y. Siret. Overview of EAT, a System for Effective Homology Computation. *The SAC Newsletter*, (3):69–79, 1998.

[106] J.J.M.M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249(1):3–80, 2000.

[107] D.E. Rydeheard y R.M. Burstall. *Computational Category Theory.* Prentice Hall, 1988.

[108] D. Sannella y A. Tarlecki. Toward formal development of programs from algebraic specifications: implementations revisited. *Acta Informatica*, 25:233–281, 1988.

[109] D. Sannella y A. Tarlecki. Essential concepts of algebraic specification and program development. *Formal Aspect of Computing*, 9:229–269, 1997.

[110] D. Sannella y A. Tarlecki. Algebraic preliminaries. En E. Astesiano, H.-J. Kreowski y B. Krieg-Brückner, editores, *Algebraic Foundations of Systems Specification*, pp. 13–30. Springer, 1999.

[111] D. Sannella y M. Wirsing. Implementation of Parameterised Specifications. En M. Nielsen y E.M. Schmidt, editores, *Proceedings of the 9th International Colloquium on Automata, Languages and Programming,* Lecture Notes in Computer Science, vol. 140, pp. 473–488. Springer, 1982.

[112] F. Sergeraert. Homologie effective. *Comptes-Rendus l'Académie des Sciences*, 304:279–282 y 319–321, 1987.

[113] F. Sergeraert. Functional coding and effective homology. *Astérisque*, 192:57–67, 1990.

[114] F. Sergeraert. The computability problem in Algebraic Topology. *Advances in Mathematics*, 104:1–29, 1994.

[115] G. Steele. *Common Lisp. The language.* Digital Press, 1984.

[116] G. Steele. *Common Lisp. The language. Second Edition.* Digital Press, 1990.

[117] A. Tarlecki. Institutions: An Abstract Framework for Formal Specifications. En E. Astesiano, H.-J. Kreowski y B. Krieg-Brückner, editores, *Algebraic Foundations of Systems Specification*, pp. 105–130. Springer, 1999.

[118] A. Tarlecki, R.M. Burstall y J.A. Goguen. Some fundamental algebraic tools for the semantics of computation: Part 3. "Indexed categories". *Theoretical Computer Science*, 91(2):239–264, 1991.

[119] S. Thompson. *Type theory and functional programming.* Addison Wesley, 1991.

[120] O. Veblen. *Analysis Situs.* AMS Collection, 1931.

[121] M. Wand. Final Algebra Semantics and Data Type Extensions. *Journal of Computer and System Sciences*, 19:27–44, 1979.

[122] W. Wechler. *Universal Algebra for Computer Scientists.* Springer-Verlag, 1992.

[123] M. Wirsing. Algebraic Specification. En J.V. Leeuwen, editor, *Formal models and semantics*, pp. 677–788. Elsevier, 1992.

[124] S.N. Zilles. *An introduction to data algebras.* Working darft paper, IBM Research, 1975.